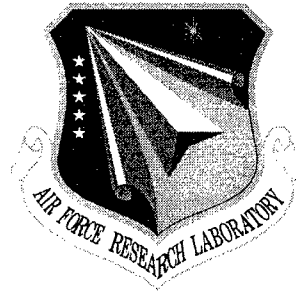


AFRL-IF-RS-TR-2001-168
Final Technical Report
August 2001



“THE INTELLIGENT ROOM”

MIT AI Laboratory

**H. Shrobe, M. Coen, K. Wilson, L. Weisman, K. Thomas, M. Groh, B. Phillips,
S. Peters, N. Warshawsky, and P. Finin**


APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

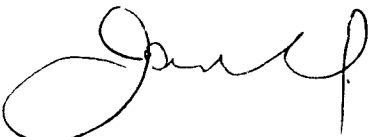
20011109 067

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-168 has been reviewed and is approved for publication.

APPROVED: 
ROBERT M. FLO
Project Engineer

FOR THE DIRECTOR: 
JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFSB, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE AUGUST 2001	3. REPORT TYPE AND DATES COVERED Final Jun 94 - Aug 99		
4. TITLE AND SUBTITLE "THE INTELLIGENT ROOM"		5. FUNDING NUMBERS C - F30602-94-C-0204 PE - 61101E PR - B322 TA - 00 WU - 01		
6. AUTHOR(S) H. Shrobe, M. Coen, K. Wilson, L. Weisman, K. Thomas, M. Groh, B. Phillips, S. Peters, N. Warshawsky, and P. Finin				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) MIT AI Laboratory 545 Technology Square Boston Massachusetts 02139		8. PERFORMING ORGANIZATION REPORT NUMBER N/A		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSB 525 Brooks Road Rome New York 13441-4505		10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2001-168		
11. SUPPLEMENTARY NOTES Air Force Research Laboratory Project Engineer: Robert M. Flo/IFSB/(315) 330-2334				
12a. DISTRIBUTION AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) Intelligent, interactive environments promise to drastically change our everyday lives by connecting computation to ordinary, human-level events happening in the real world. The motivation for building these intelligent, interactive environments is to bring computation into the real, physical world to support what are traditionally considered non-computational activities. The ultimate goal is to allow people to interact with computational systems in a way that they would with other people: via gesture, voice, movement, and context. An existing prototype space, known as The Intelligent Room, was created to experiment with different forms of natural, multi-modal human-computer interaction. This research platform is equipped with numerous computer vision, speech, and gesture recognition systems that connect it to what the inhabitants of the room are doing and saying. The purpose of this report is to detail the design of The Intelligent Room and the experiences gained during its creation.				
14. SUBJECT TERMS Human-Computer Interactions, Intelligent, Interactive Interfaces/Environments, Multi-Modal Interaction			15. NUMBER OF PAGES 56	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

Introduction	1
Building Brains for Rooms: Designing Distributed Software Agents	2
Design Principles for Intelligent Environments	9
Learning Spatial Event Models from Multiple-Camera Perspectives	17
A Context Sensitive Natural Language Modality for an Intelligent Room	26
Meeting the Computational Needs of Intelligent Environments: The Metaglu System	37

Introduction

This report consists principally of five research papers that cover the extent of the Intelligent Room Project carried out at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. They cover a five-year period of work, and illustrate the evolution of the technologies developed in support of this project. The papers are:

1. Building Brains for Rooms: Designing Distributed Software Agents, by Michael H. Coen
2. Design Principles for Intelligent Environments, by Michael H. Coen
3. Learning Spatial Event Models from Multiple-Camera Perspectives, by Michael H. Coen and Kevin W. Wilson
4. A Context Sensitive Natural Language Modality for an Intelligent Room, by Michael Coen, Luke Weisman, Kavita Thomas, and Marion Groh
5. Meeting the Computational Needs of Intelligent Environments: The Metagluue System, by Michael H. Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin

The first two of these discuss the overall principles of organization in the Intelligent Room; the third treats the specific problems of the use of visual images in the Room; the fourth deals with linguistic interactions with the Room, spoken and written; and the last describes the eventual computational infrastructure that binds the Room together as a whole.

Building Brains for Rooms: Designing Distributed Software Agents

Michael H. Coen

MIT AI Lab
545 Technology Square
Cambridge, MA 02139
mhcoen@ ai.mit.edu

Abstract

This paper argues that complex, embedded software agent systems are best constructed with parallel, layered architectures. These systems resemble Minskian Societies of Mind and Brooksian subsumption controllers for robots, and they demonstrate that complex behaviors can be had via the aggregates of relatively simple interacting agents. We illustrate this principle with a distributed software agent system that controls the behavior of our laboratory's Intelligent Room.

Introduction

This paper argues that software agent systems that interact with dynamic and complex worlds are best constructed with parallel, layered architectures. We draw on Brooks' subsumption architecture (Brooks, 1985) and Minsky's Society of Mind (Minsky, 1986) theory to dispel the notion that sophisticated and highly capable agent systems need elaborately complex and centralized control.

Towards this end, we present an implemented system of software agents that forms the backbone of our laboratory's "Intelligent Room" (Torrance, 1995). These agents, known collectively as the *Scatterbrain*, control an environment very tenuously analogous to the intelligent rooms so familiar to Star Trek viewers --- i. e., rooms that listen to you and watch what you do; rooms you can speak with, gesture to, and interact with in other complex ways.

The Scatterbrain consists of approximately 20 distinct, intercommunicating software agents that run on ten different networked workstations. These agents' primary task is to link various components of the room (e. g., tracking cameras, speech recognition systems) and to connect them to internal and external stores of information (e. g., a person locator, the World Wide Web). Although an

individual agent may in fact perform a good deal of computation, we will focus our interest on the ways in which agents get connected and share information rather than how they internally manipulate their own data. And while the Intelligent Room is a fascinating project in itself, we will treat it here mainly as a test-bed to learn more about how software agents can interact with other computational and real entities.

Our approach has also been modeled on a somewhat unorthodox combination of the Brooks (Brooks, 1991) and Minsky approaches to core AI research. As pointed out in (Kautz et. al., 1994), it is difficult to find specific tasks for individual agents that are both feasible and useful given current technology. Many of the non-trivial tasks we would like software agents to perform are simply beyond the current state of the art. However, taking our cue from Minsky, we realize interesting and complex behaviors can be had via the aggregates of simpler ones: groups of simple agents can be combined to do interesting things. We also found Brooks' subsumption architecture useful for guiding the creation of the Scatterbrain, particularly for building parallel layers of behaviors that allow the room to process multiple events simultaneously and to change contexts quickly. In many ways, the room is similar to a disembodied robot, so it comes as no surprise that the robotics community can provide insight into how the room's brain should be designed. We argue, however, that this does not preclude insights obtained in creating the Scatterbrain from applying to other distributed software agents systems. Rather, as argued by Etzioni (Etzioni, 1994, 1996; Etzioni et al. 1994), even agents who solely interact with the online world (and don't have cameras for eyes and microphones for ears) can be viewed as a kind of simulated, virtual robot. More important than its connection with the real world, what the Scatterbrain shares with Brooks' robots is its organizational structure and its lack of central processing; all of the Scatterbrain's agents work together in parallel with different inputs and data being processed simultaneously in different places.

The next section of this paper describes the Intelligent Room's physical infrastructure. After this, we introduce the room's most recent application, a *Tour Guide* agent that helps a person present our lab's research to visitors.

¹ Copyright © 1997, American Association for Artificial Intelligence (www. aaai. org). All rights reserved.

² This material is based upon work supported by the Advanced Research Projects Agency of the Department of Defense under contract number F30602—94—C—0204, monitored through Rome Laboratory and Griffiss Air Force Base.

Next, we present in detail the room's software agent architecture, including the design and implementation of several components of the Scatterbrain and Tour Guide agents. We also contrast our approach with several earlier monolithic efforts taken in our lab to program and control the behavior of the Intelligent Room.

Part of the motivation for this work has been to push the envelope of software agent design. Much has been made over the lack of obvious "killer applications" for software agents. After all, how many automated meeting schedulers does the world need? We are interested in exploring new realms of complex interactions for software agents which in and of themselves constitute these "killer apps" that have been seemingly so elusive from the single-agent perspective. Minsky argues that societies of agents can do things that seem inordinately complex when their behavior is viewed as the work of a single entity. Our experiments with fairly large assemblies of software agents mark an early attempt towards establishing that this is indeed the case.

The Intelligent Room

The Intelligent Room project explores new forms of interaction and collaboration between people and computers. Our objective is to create a new kind of environment capable of interpreting and contributing to activity within it. On a grand scale, we are examining a paradigmatic shift in what it means to use a computer. Rather than view a computer as a box with a keyboard and monitor used for traditional computational tasks, we envision computers being embedded in the environment and assisting with ordinary, traditionally non-computational activity. For example, if I lose my keys in the Intelligent Room, I'd someday simply like to ask the room where I left them.

The Intelligent Room is an excellent environment in which to conduct core AI research according to the criteria of both Brooks (Brooks, 1991) and Etzioni (Etzioni, 1994). The room is "physically" grounded in the real-world. The room's cameras and microphones allow it to deal with the kinds of complex, unpredictable and genuine phenomena that Brooks argues is essential for a core AI research testbed. However, the room also processes abstract, symbolic information that actually represents something extant, thereby satisfying Etzioni's desiderata. For example, if a person asks the room, *What is the weather in Boston?*, the room needs to recognize more than a meaningless *weather* token -- it needs to get that information and display it to the user. This is done using a variety of information retrieval systems available on the World Wide Web.

This section first describes the room's physical infrastructure. We then present the room's most recent application, a *Tour Guide* agent that helps a person present our lab's research to visitors. In the next section, we discuss in detail the room's software agent architecture, including the design and implementation of the

Scatterbrain and Tour Guide agents.

Infrastructure - From the bottom up

Figure 1 diagrams the room's physical layout. The Intelligent Room's infrastructure consists of several hardware and software systems that allow the room to observe and control its environment in real-time. The positions of people in the room are determined using a multi-person tracking system. Hand pointing actions are recognized by two separate gesture recognition systems. The one used in the application described below allows the room to determine where someone is pointing on either of two images projected on a room wall from high-intensity, ceiling mounted VGA projectors. A speech recognition system developed by (Zue, 1994) allows the room to listen to its inhabitants, and it is used in conjunction with a speech generator to enable the room to engage in sustained dialogues with people. The room interfaces with the START natural-language information retrieval system (Katz, 1990) to enhance its ability to understand complex linguistic input. The room also controls two VCRs and several other video displays in addition to the ceiling mounted projectors. A matrix switcher allows arbitrary connections between the room's audio/ visual inputs and outputs.

The room's hardware systems are directly interfaced with low-level C programs to insure their real-time operation. For example, the room's tracking cameras have 30 Hz frame rates and their data streams need to be synchronously processed using direct operating system calls.

Figure 1 - Intelligent Room Floor Plan

The Tour Guide Agent

The room's most recent application provides support for someone giving tours of our laboratory. These tours typically involve a group of visitors meeting with a graduate student who discusses and answers questions about the lab's research and shows several video clips. Rather than have these presentations given in an ordinary

conference room, we have decided to have them in the Intelligent Room so the room can assist the human tour guide. A typical dialogue between the room and student tour guide is:

Tour guide : *Computer, load the AI Lab tour*

Room : *I am loading the AI Lab tour.* Right projector now displays a Netscape browser window with a special Lab Tour home page

Tour guide : Using hand, points at link to a research project displayed on the wall and says, *Computer, follow this link*

Room : Loads the indicated page into the browser.

Tour guide : *Computer, show me the Intelligent Room home page.*

Room : Loads the URL corresponding to the name of the page. Then says, *I have a video clip for this research. Would you like to see it?*

Tour guide : *Computer, yes.*

Room : Moves appropriate video cassette to correct position and starts the clip playing on left projector.

Tour guide: (watches video for a few seconds) *Computer, stop the video. Computer, play the Virtual Surgery clip.*

Room : Performs requested action. Stops video when clip is done.

Tour guide : *Computer, how many graduate students are there at the AI Lab?*

Room : *I am asking the START system for the answer ... The Laboratory's 178 members include 17 faculty members, 26 academic staff, 29 research and support staff, and 106 graduate students.* Also displays web page with elaborated answer.

Other applications include a control center for planning hurricane disaster relief and an intelligent living room.

Control Architectures

The room has been discussed so far at its most concrete and most abstract: namely, its hardware infrastructure and its high-level software applications. How these applications are actually created on top of this infrastructure, i. e., how the room actually works, is the subject of this section.

Monolithic Control

In its early stages of development, each of the room's components was wrapped inside a TCP client or server that connected with a monolithic C-language program that controlled the room's behavior. Figure 2 contains this controller along with each of the programs it connected with. (Included in parentheses with each component is the name of the workstation it ran on.)

From a conceptual point of view, the most serious flaw with the centralized controller was that it failed to distinguish between basic functioning common to all room contexts – such as noticing when someone came through the doorway – and unique activities associated with a particular room application. Furthermore, adding new functionality to the room required modifying the monolithic controller and manually determining the interactions and conflicts between old and new room

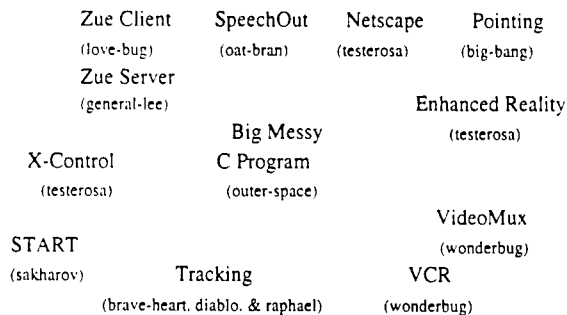


Figure 2 - The Monolithic Controller

functions. There was no way to modularly add new room capabilities on top of old ones and assume everything would continue working as expected.

Also, directing the information flow among the room's various components—one of the main functions of the controller – was overly difficult in a language like C. We needed higher-level mechanisms for describing how room information moved among its producers and consumers.

From a practical point of view, the monolithic controller also made it difficult to reconfigure the room dynamically or restart pieces of the room independently of others. We often found while working on the room that in order to restart one component, it was necessary to restart the entire room. This was particularly frustrating because starting the room required the coordinated activity of several people to start particular programs (in a predetermined order) and configure various room hardware. It was also difficult to move components of the room to different workstations because that required modifying hard-coded machine dependencies in the code.

SodaBot

Although we managed to use the monolithic approach for several very simple applications, it seemed unlikely to scale to the more complex interactions we had in mind for the room. Our initial dissatisfaction with this architecture led to the adoption of the SodaBot software agent platform (Coen, 1994) for duplicating the functionality of our initial monolithic room controller with a system of distributed software agents.

SodaBot provides both a programming language and runtime platform for software agents, and it simplifies writing interactive programs by providing high-level primitives for directing flows of online information. For example, it provides mechanisms for writing agent-wrappers that interface with preexisting software either via text-based or graphical user interfaces (X-windows and Windows 95/ NT).

For example, we created a SodaBot *Netscape Agent* that controls interactions with a Netscape browser. It offers functions to other agents such as those listed below.

Function	Purpose
New (host)	Runs a new browser on given host
Load(url)	Loads URL in browser
Page_ watch()	Arranges for notification (of URL) to another agent whenever browser loads new page
LaserPointing	
Link_ watch()	Arranges for notification to another agent when a new page is loaded containing its URL/ anchor text pairs
Text	Returns text of current page
Page(direction)	Moves browser scroll-bar in given direction

For the Intelligent Room, we use SodaBot agents as *computational glue* for interconnecting all of the room's components and moving information among them. Initially, we simply duplicated the room's monolithic controller using SodaBot's high-level programming language. Most notably, SodaBot simplified description of room functioning and interaction with remote TCP-based clients and servers by removing networking and hardware details. However, this new room controller, dubbed the *Brain*, was still a computational bottleneck, and we had yet to distinguish between a general behavioral infrastructure for the room (i. e. its core functionality) and the more complex, application specific interactions we built on top of it. This led to the development of the room's current control system, the Scatterbrain, which is the subject of the next section.

Distributed Room Control

The Scatterbrain (Figure 3) is platform on top of which room applications can be layered. In the figure, each circle represents a distinct SodaBot software agent that is wrapped around and interfaced with an external application. (The layer containing these "base applications" is not shown.) Each of the Scatterbrain agents is responsible for a different room function. For example, the *SpeechIn Agent*, runs and interfaces with our speech recognition systems. Once started, *SpeechIn* allows other agents to submit context-free grammars corresponding to spoken utterances they are interested in. As they do, it updates the speech recognition systems to contain the current set of valid utterances. When a sentence is heard by one of the speech systems, *SpeechIn* then notifies those agents who indicated they were interested in it. As another example, the *Netscape Agent* connects to the *Display Agent* to make sure that when web pages are loaded, the browser is actually displayed somewhere in the room where people can see.

The Scatterbrain agents are distributed among 10 different workstations and rely on SodaBot interagent communication primitives to locate and communicate with each other. The lines in the figure represent default interactions the room manifests in all applications, such as having various agents connect with the speech recognition agents and making sure the tracking system notices when they connect to non-displayed external applications. The

someone comes in the room. Essentially, the Scatterbrain implements the Intelligent Room's reflexes.

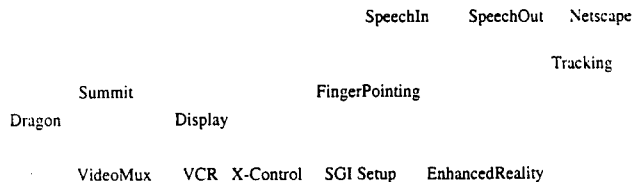


Figure 3 – The Agents of the Scatterbrain

The room no longer has a central controller. A small startup agent runs all of the Scatterbrain agents which then autonomously move to the machines on which they are supposed to run. All the Scatterbrain agents then work together in parallel with different inputs and data being processed simultaneously in different places. This makes the room robust to failure in one of its sub-systems and allows us to restart sections of the room independently. Also, the SodaBot system allows real-time data connections between agents to be broken and resumed invisibly. For example, if the *Tracking Agent* is updating another agent in real-time, either one of them can be stopped and restarted and they will be automatically reconnected.

Layered on top of the Scatterbrain, we created higher-level agents that rely on the Scatterbrain's underlying behaviors. Figure 4 contains the room's intermediate information-level applications such as a *Weather Agent* that can obtain forecasts and satellite maps for particular places. By relying on the previously described interaction, if the Weather Agent uses the Netscape Agent to display information, it doesn't need to be concerned with insuring the browser is displayed in a place where the user is looking.

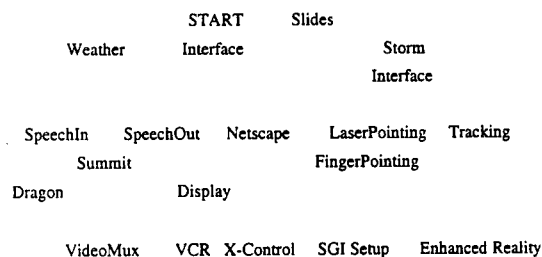


Figure 4 – Information Agents

We then created specific room application agents that relied on the lower-level, general-purpose agents in the room. Figure 5 contains a diagram of several room applications and how they connect to the room's underlying architecture. Note that all of the objects in the figure represent SodaBot software agents and many of called the *X-Server* that controls the actual SGI workstation

next section explores two of the application agent interactions in more detail.

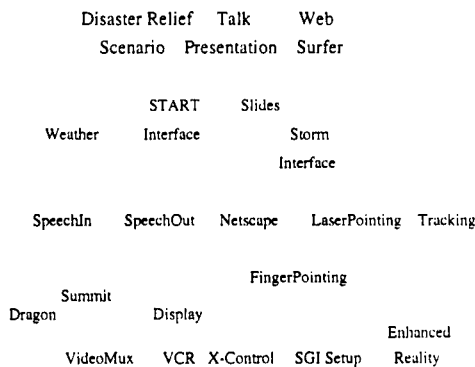


Figure 5 – Intelligent Room Software Agents

Agent Interaction

This section examines how we can get the room to exhibit interesting behavior by layering agents on top of each other. We examine two separate room behaviors and then discuss how they combine to produce greater functionality.

We have a system in the room called *Storm*, used in a disaster relief planning scenario, that can display scalable maps of the Caribbean. People can interact with Storm using pointing and speech. For example,

User: *Computer, display Storm on the left projector.*

(User now points at Puerto Rico.)

User: *Computer, zoom in.*

(User now points at San Juan.)

User: *Computer, what is the weather here?*

(The room then displays a weather forecast for San Juan inside a Netscape browser on the other projector.)

To see how this scenario works, we first examine pointing recognition as an example of simple agent interaction. We then look at a more complex scenario from the Tour Guide agent presented earlier.

By default, the room's projectors are set by the *Display Agent* to show portions of the screens of two of our SGI workstations. If someone points someplace close to one of these projected displays, the display's mouse cursor moves

to that position. Although this seems like a trivial process, there is a fair amount of effort behind it, as shown in Figure 6. The person moving his finger is reflected in the camera images received by the neural network pointing software. This reconciles the images to produce new pointing information. These new data are passed to the *FingerPointing Agent* which is responsible for handling all such events in the room. By default, the Scatterbrain has all pointing events on the each display sent to an agent

generating the display. This X-Server agent then moves the mouse cursor to the appropriate position, which is reflected in the displayed image. However, the Storm Agent overrides this default behavior and redirects pointing events on the Storm display to itself. Upon receipt of a pointing event, it updates the Storm application's internal cursor, which moves intelligently between salient geographical features. For example, pointing near San Juan will cause the Storm program to register the city with the Storm Agent, rather than a point three pixels to its left. Finally, note that the various agents are responsible for translating between the room's many coordinate systems, as shown along the connections.

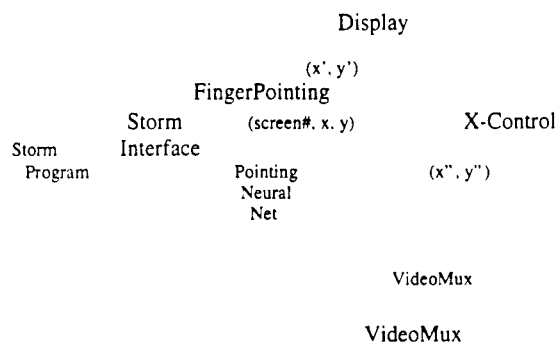


Figure 6 – Pointing in the Room

When someone in the room says *What's the weather here?*, the *SpeechIn Agent* notifies the room's *Disaster Relief Planning Agent* because this utterance is contained in the grammar that agent had registered when first run. The *Storm Agent* is then contacted to determine what geographical entity is closest to where the person was pointing close to the time they asked the question. (Low-level room events are time-stamped by agents in order to facilitate multimodal reconciliation.) This process is shown in Figure 7.

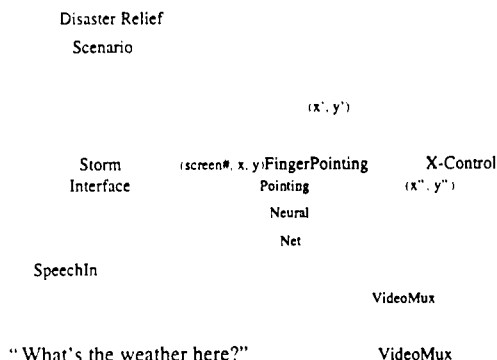


Figure 7 – Multimodal Resolution

Conclusion

Motivated by Minsky's Society of Mind and Brooks' subsumption approach to building robots, we have argued that software agent systems that interact with complex and dynamic worlds are best created from distributed collections of simple agents with parallel, layered architectures.

The complexity of the overall system comes from the interactions of these agents, even though no individual agent is in itself particularly complex and no single agent centralizes the system's control. This approach allows us to build robust, reusable, and behaviorally sophisticated systems that are capable of interacting with the ever-changing real and online worlds. To demonstrate this approach, we presented the Scatterbrain – a distributed collection of software agents that control our laboratory's Intelligent Room.

Acknowledgements

Development of the Intelligent Room has involved the efforts of many people. Professors Tomas Lozano-Perez, Lynn Stein and Rodney Brooks were principally responsible for the room's conception and construction. Mark Torrance led the project during its first year and wrote the room's earliest monolithic controllers. The room's many vision systems are due to the efforts of Jeremy De Bonet, Chris Stauffer, Sajit Rao, Tomas Lozano-Perez, Darren Phi Bang Dang, JP Mellor, Gideon Stein, and Kazuyoshi Inoue. Polly Pook contributed to the design of the room's distributed computation and has worked on modeling the room's functionality as a cognitive process. Josh Kramer wrote large sections of the Scatterbrain and participated in the development of the SodaBot system. All of the above mentioned were also responsible for designing room applications, and many of the above hacked on various room components. Kavita Thomas, along with help from Mark, Polly, and Tomas, configured Victor Zue's speech recognition system. (Jim Glass provided assistance in getting the system running.) Boris Katz and Deniz Yuret provided much support in interfacing with and customizing the START natural language system. Mike Wessler created one of the room's earliest applications and wrote invaluable graphical interfaces for much of the room's hardware.

References

Brooks R. 1985: A Robust Layered Control System for a Mobile Robot, AI Lab Memo 864, Massachusetts Institute of Technology. Cambridge, MA.

Brooks R. 1991: Intelligence without Representation, in Special Volume: Foundations of Artificial Intelligence, Artificial Intelligence, 47(1-3).

Coen, M. 1994. SodaBot: A Software Agent Environment and Construction System. AI Lab Technical Report 1493. Massachusetts Institute of Technology. Cambridge, MA.

Etzioni, O. 1994: Intelligence without Robots, AI Magazine, Winter 1994.

Etzioni, O.; Levy, H.; Segal, R.; and Thekkath, C. 1994. OS Agents: Using AI Techniques in the Operating System Environment. Technical Report 93-04-04. Dept. of Computer Science. University of Washington. Seattle, WA.

Etzioni, O. 1996: Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web, in Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI Press/ MIT Press, Cambridge, MA, pp. 1322-1326, 1996.

Kautz, H.; Selman, B.; Coen, M.; and Ketchpel, S. 1994. An Experiment in the Design of Software Agents. In Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI Press/ MIT Press, Cambridge, MA.

Katz, B. 1990. Using English for Indexing and Retrieving. In *Artificial Intelligence at MIT: Expanding Frontiers*. Winston, P.; and Shellard, S. (editors). MIT Press, Cambridge, MA. Volume 1.

Minsky, M. 1986. *Society of Mind*. New York. Simon and Schuster.

Torrance, M. 1995. Advances in Human-Computer Interaction: The Intelligent Room, In Working Notes of the CHI 95 Research Symposium, May 6-7, 1995, Denver, Colorado.

Zue, V. 1994. Human Computer Interactions Using Language Based Technology, IEEE International Symposium on Speech, Image Processing & Neural Networks, Hong Kong..

Design Principles for Intelligent Environments

Michael H. Coen

MIT Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139
mhcoen@ai.mit.edu

Abstract

This paper describes design criteria for creating highly embedded, interactive spaces that we call Intelligent Environments. The motivation for building these systems is to bring computation into the real, physical world to support what is traditionally considered non-computational activity. We describe an existing prototype space, known as the Intelligent Room, which was created to experiment with different forms of natural, multimodal human-computer interaction. We discuss design decisions encountered while creating the Intelligent Room and how the experiences gained during its use have shaped the creation of its successor.

1. Introduction

This paper describes design criteria for creating highly embedded, interactive spaces that we call *Intelligent Environments* (IEs). The motivation for building IEs is to bring computation into the real, physical world. The goal is to allow computers to participate in activities that have never previously involved computation and to allow people to interact with computational systems the way they would with other people: via gesture, voice, movement, and context.

We describe an existing prototype space, known as the *Intelligent Room*, which is a research platform for exploring the design of intelligent environments. The Intelligent Room was created to experiment with different forms of natural, multimodal human-computer interaction (HCI) during what is traditionally considered non-computational activity. It is equipped with numerous computer vision, speech and gesture recognition systems that connect it to what its inhabitants are doing and saying.

Our primary concern here is how IEs should be designed and created. Intelligent environments, like

traditional multimodal user interfaces, are integrations of methods and systems from a wide array of subdisciplines in the Artificial Intelligence (AI) community. Selecting the modal components of an IE requires a careful strategic approach because of the *a priori* assumption that the IE is actually going to be embedded in the real-world. In particular, there is a need for the use of synergy (Cohen [4]) to allow imperfect modalities to reinforce and support each other.

We discuss below the design of our laboratory's Intelligent Room and how experiences gained during its use have shaped the creation of its successor. Given the increasingly widespread interest in highly interactive, computational environments (Bobick et al. [3]), (Coen [6,7,8]), (Cooperstock et al. [10]), (Lucente et al. [17]), we hope these experiences will prove useful to other IE designers and implementers in the AI community.

Some of the earliest work in this area has been done wholly outside the AI community. This is primarily due to the perception that AI has little to offer in the way of robust, ready for the real world systems. We contend that Intelligent Environments not only would benefit from AI subdisciplines ranging from knowledge representation to computer vision, but they would be severely limited without them.

Outline

Section 2 describes some sample interactions with and applications of the Intelligent Room. These range from an intelligent command post to a reactive living room. Comparison to other HCI paradigms, such as ubiquitous computing, and other embedded computational environments is contained in section 3. Section 4 presents the Intelligent Room's physical infrastructure. Sections 5 and 6 detail the Intelligent Room's visual and spoken language modalities. We document the rationales that influenced our approach, system limitations, and solutions we are pursuing in the development of the next generation Intelligent Room currently under construction in our laboratory.

2. Room Interactions

Our approach with the Intelligent Room has been to create a platform for HCI research that connects with real-

This material is based upon work supported by the Advanced Research Projects Agency of the Department of Defense under contract number F30602-94-C-0204, monitored through Rome Laboratory. Additional support was provided by the Mitsubishi Electronic Research Laboratories.

Copyright © 1998, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

world phenomena through several computer vision and speech recognition systems. These allow the room to watch where people are moving, under certain circumstances where they are pointing, and to listen to a fairly wide variety of spoken language utterances.

The Intelligent Room supports a variety of application domains. One of these is a command center for planning hurricane disaster relief in the Caribbean. This makes use of two interactive projected displays that respond to both finger pointing and laser pointing gestures. A sample interaction with the disaster relief center is:

User: "Computer, <pause> stay awake."

[The room will now listen for utterances without requiring they be prefaced by the word *Computer*.]

User: "Show me the Virgin Islands."

Room: "I'm showing the map right next to you." [Room shows map on video display closest to the user.]

User: [now points at St. Thomas.] "Zoom in. How far away is Hurricane Marilyn?"

Room: "The distance between Hurricane Marilyn and the city of Charlotte Amalie located in St. Thomas is 145 miles."

User: "Where's the nearest disaster field office?"

[Room highlights them on the map.]

Room: "The St. Thomas disaster field office is located one mile outside of Charlotte Amalie. Michael, there is a new weather forecast available. Do you want to see it?"

User: "Yes, show me the satellite image."

We are currently developing a next generation of the Intelligent Room, called *Hal* (after the computer in the movie, *2001: A space odyssey*). *Hal* is furnished like a combination home/office and supports a wider range of activities than the original Intelligent Room. A scenario that currently runs within *Hal* is:

I walk into *Hal* and lie down on the sofa after shutting the door. *Hal* sees this, dims the lights, closes the curtains, and then puts on Mozart softly in the background. *Hal* then asks, "*Michael, what time would you like to get up?*"

The goal of implementing these types of scenarios is to explore and help define what an intelligent environment should be, what sensory capabilities it needs, and to determine what roles such environments could potentially play in our lives. In the process, these scenarios provide insight into both how AI systems can participate in the real world and directions for further research in the subdisciplines whose systems contribute to the creation of intelligent environments.

3. Motivation

Intelligent environments are spaces in which computation is seamlessly used to enhance ordinary activity. One of the driving forces behind the emerging interest in highly

interactive environments is to make computers not only genuinely user-friendly but also essentially invisible to the user. The user-interface primitives of these systems are not menus, mice and windows but gesture, speech, affect, and context. Their applications are not spreadsheets and word processing but intelligent rooms and personal assistants.

Intelligent environments are both embedded and multimodal and thereby allow people to interact with them in natural ways. By being embedded, we mean these systems use cameras for eyes, microphones for ears, and ever-increasingly a wide-range of sophisticated sensing technologies to connect with real-world phenomena. Computer vision and speech recognition/understanding technologies can then allow these systems to become fluent in natural forms of human communication. People speak, gesture, and move around when they communicate. For example, by embedding user-interfaces this way, the fact that people tend to point at what they are speaking about is no longer meaningless from a computational viewpoint and we can build systems that make use of this information. In some sense, rather than make computer-interfaces for people, we want to make people-interfaces for computers.

Coupled with their natural interfaces is the expectation that these systems are not only highly interactive (i.e. they talk back when spoken to) but also that they are useful for ordinary activities. They should enable tasks historically outside the normal range of human-computer interaction by connecting computers to phenomena (such as someone walking into a room) that have traditionally been outside the purview of contemporary user-interfaces.

Why this isn't Ubiquitous Computing

Intelligent environments require a highly embedded computational infrastructure; they need many connections with the real world in order to participate in it. However, this does not imply that computation need be everywhere in the environment nor that people must directly interact with any kind of computational device. Our approach is to advocate minimal hardware modifications and "decorations" (e.g., cameras and microphones) in ordinary spaces to enable the types of interactions in which we are interested. Rather than use the computer-everywhere model of ubiquitous computing – where for example, chairs have pressure sensors that can register people sitting in them or people wear infrared-emitting badges so they can be located in a building – we want to enable unencumbered interaction with non-augmented, non-computational objects (like chairs) and to do so without requiring that people attach high-tech gadgetry to their bodies (as opposed to the approach in [24,25]).

AI-based approaches have much to offer these environments. For example, although a pressure sensor on a chair may be able to register that someone has sat down, it is unlikely to provide other information about that person, e.g., her identity. Visual data from a single

camera can provide far more information than simple sensing technologies. This includes the person's identity, position, gaze direction, facial expression, gesture, and activity ([13, 25,17,30,12]). While there has yet to be a coherent system that unifies all of these capabilities, many prototypes are currently under development. Furthermore, enhancing the capabilities of a computer vision system often requires modifying only the software algorithms that process incoming images and not the room's sensory components. Also, because the room senses at a distance, objects, in particular people and furniture, do not need to be physically augmented and/or wired for the room to become aware of them.

Other related work

The DigitalDesk project (Wellner [26], Newman et al. [19]) was an early and influential system that had a bird's eye view of a desktop through an overhead video camera. It recognized and responded to predetermined hand gestures made by users while interacting with real paper documents on the surface of a desk. The Intelligent Room has a desktop environment directly motivated by the DigitalDesk, which recognizes a wider range of complex hand gestures (Dang [11]).

Other substantial efforts towards highly interactive environments include an automated teleconferencing office (Cooperstock et al. [10]) and an immersive fictional theater (Bobick et al. [3]). Each of these projects makes use of embedded computation to enable unusual human-computer interactions, e.g., vision-based person tracking. However their modal processing is extraordinarily specific to their applications, and the applicability of such carefully tuned systems to other domains is unclear. The Classroom 2000 project (Abowd et al. [1]) is an educational environment that automatically creates records linking simultaneous streams of information, e.g. what the teacher is saying while a student is writing down her notes on a digital pad. Mozer ([18])

describes a house that automatically controls basic residential comfort systems, such as heating and ventilation, by learning patterns in its occupants behavior.

Related user-interface work such as Cohen et al. [5] uses multimodal interface technology to facilitate human interaction with a preexisting distributed simulator. In doing so, it provides a novel user-interface to a complex software system, but it is one that requires tying down the user to a particular computer and a specific application. We are interested in creating new environments that support never before conceived of applications – applications that historically have not involved computation.

4. The Intelligent Room

The Intelligent Room occupies a 27'x37' room in our laboratory. Approximately half of this space is laid out like an ordinary conference room, with a large table surrounded by chairs. (See Figure 1.) This section has

two bright, overhead LCD projectors in addition to several video displays. There is also an array of computer controlled video equipment which is discussed below.

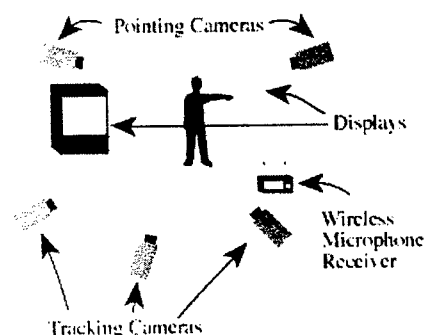


Figure 1 – A skeletal view of the conference area in the Intelligent Room

Mounted at various places in the conference area are twelve video cameras, which are used by computer vision systems.

Separated from the conference area by a small partition and occupying the rest of the room are most of the workstations that perform the room's computation. The section of the room is not interactive, but having it adjacent to the interactive conference area simplifies wiring, implementation and debugging.

The Intelligent Room contains an array of computer controlled devices. These include steerable video cameras, VCRs, LCD projectors, lights, curtains, video/SVGA multiplexers, an audio stereo system, and a scrollable LCD sign. The room's lighting is controlled through several serially interfaced X-10 systems. Many of the room's other devices have serial ports that provide both low-level control and status information, e.g., our VCRs can report their present position on a videotape to give us random access to video clips. The room can also generate infrared remote control signals to access consumer electronics items (namely, objects that don't have serial ports).

Room Controller

When the Intelligent Room was in the early stages of its design and construction, the most challenging research problems appeared to be developing its computer vision and speech recognition/understanding systems. What was not obvious is that interconnecting all of the room's many subsystems and coordinating the flows of information among the room components was a non-trivial problem. Developing a software architecture that allowed the room to run in real-time and cope with vagaries of its real-world interactions emerged to be one of the room's chief research problems.

What emerged from an iterative development process is a modular system of software agents known collectively

as the *Scatterbrain* (described in detail in Coen [6]). The Scatterbrain currently consists of approximately 50 distinct, intercommunicating software agents that run on ten different networked workstations. These agents' primary task is to connect various components of the room (e.g., tracking and speech recognition systems) to each other and to internal and external stores of information (e.g., a person locator or an information retrieval system). Essentially, the Scatterbrain agents are intelligent *computational glue* for interconnecting all of the room's components and moving information among them.

5. Room Vision Systems

Person Tracking

The Intelligent Room can track up to four people moving in the conference area of the room at up to 15Hz. The room's person tracking system (DeBonet [13]) uses two wall-mounted cameras, each approximately 8' from the ground. (A debugging window from the system showing the view from one of the cameras is shown in Figure 2.)

We initially decided that incorporating a tracking system in the Intelligent Room was essential for a number of reasons. It gives the room the ability to know where and how many people are inside it, including when people enter or exit. The room is able to determine what objects people are next to, so for example, it can show data on a video display someone is near. A person's location in the room also provides information about what she is doing. For example, someone moving near a video display while others are seated around the conference table might indicate she is giving a presentation.

The tracking data are useful for supplying information to other systems in the room including, to our surprise, our speech understanding system. It was clear from the start that tracking could disambiguate other room modalities, for example, by providing a foveal area for gesture recognition. However, its use in providing contextual information to the room's speech recognizer is a revealing example of how one modality can be used to help overcome the weaknesses of another. In this case, where people are in the room can sometimes provide information about what they are likely to say (see section 6).

The tracking system works via background segmentation and does 3D reconstruction through a neural network. The output image from each camera is analyzed by a program that labels and identifies a bounding box around each occupant in the room. This information is then sent through a coordination program that synchronizes the findings from the individual cameras and combines their output using a neural network to recover a 3D position for each room occupant. People are differentiated in the system using color histograms of their clothing, which the room builds when they first

come inside. Because the room's configuration is fairly static and the cameras used for tracking are stationary, the tracking system can build a model of the room's relatively

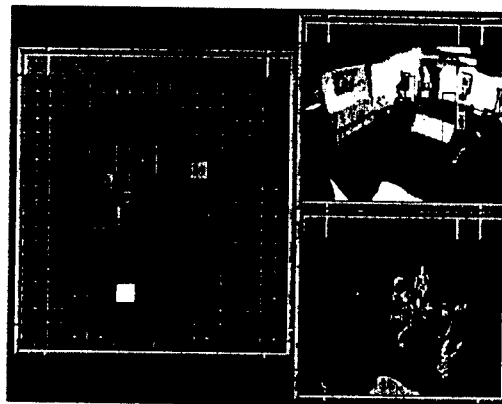


Figure 2 - Tracking System Debug Window

slowly changing background to compare with incoming images.

The tracking subsystem also controls three steerable cameras. These can be used to follow individuals as they move about the room or to select optimal views of people given their position and previous knowledge of room geometry, e.g. where people likely face when standing in particular areas of the room.

This approach differs from the overhead tracking system described in Bobick et al. [3]. Their domain had 27' high ceilings, for which it is quite reasonable to look for people from a single camera bird's eye perspective. Rooms with ordinary height ceilings do not make this possible, so a stereo vision system seems necessary for performing background segmentation.

Pointing

The Intelligent Room's two overhead LCD video projectors display next to each other on one of the room's walls. Each can display SVGA output from one of the room's workstations or composite signals from any of the room's video sources, e.g., a VCR. These projected displays support both finger and laser pointing interactions. For example, the room can track the finger of a person who is pointing within four inches of the wall where images are displayed. Alternatively, the person can use a laser pointer to interact with the display from a distance. Both of these pointing systems also allow displayed screen objects to be selected (i.e. clicked) or moved (i.e. dragged).

Additionally, the pointing systems allow people to treat the displays like virtual whiteboards. The room can draw a visible trail on top of a displayed image that follows the continuous path of a motile pointing gesture. This allows people to overlay handwritten text and drawings on top of whatever information the room is displaying. These can then be automatically recalled at a later date, for example,

when the room shows this information again.

The finger pointing system uses two cameras mounted parallel to the wall on either side of the displays. It makes use of only three scan lines from each camera image to explore the region closest to the wall's surface. The laser pointing system uses a camera roughly orthogonal to the plane of the display wall to locate the laser's distinctive signature on the wall's surface. These systems run at approximately 15-20Hz, depending on the precise type of interaction, and provide resolution per display ranging from approximately 640x480 for laser pointing to 160x120 for finger pointing. Although the pointing systems are sufficiently responsive for discrete pointing and dragging events, handwriting recognition using the above mentioned drawing feature does not seem practical without at least doubling the sampling frequency.

Interactive Table

Through a ceiling mounted camera, the room can detect hand-pointing gestures and newly placed documents on the surface of the conference table. The gesture recognition system has been used to support a wide variety of functions (described in Dang [11]). We found, however, that making gestures over the surface of a table was not a particularly natural form of interaction and required extensive practice to master. As has been widely observed in the graphical user interface community, we found that increased novelty in an interface does not necessarily lead to increased utility. This is even more pertinent in domains like the Intelligent Room, which stress natural modes of interaction.

One useful application of this system, however, allows people to place Post-It™ notes on the surface of the table and assign to them particular functions, such as dimming the lights or announcing the current time. Touching a given note then evokes its assigned behavior from the Intelligent Room. As a mnemonic, the function of each note can be handwritten upon it, giving the table the feeling of a virtual, very non-standard control panel. The room is oblivious to any written information on these notes, as long as it doesn't interfere with the color segmentation that allows individual notes to be recognized.

Issues

Our person tracking system uses a neural network to perform 3D reconstruction. The tracking network is trained by laying a masking tape coordinate system on the room's floor and then training the network by having a person stand at each intersection of the axes. (The grid was roughly 10x20.) Although non-network approaches to 3D reconstruction are possible, such as directly calculating the reverse projective transformation, they would all likely require a user-intensive preliminary learning period to determine the transformation between room and image space. Thus, installing our tracking system is labor intensive and requires some familiarity

with how it operates.

Another difficulty with this approach is that the system is enormously sensitive to any deviation from its training conditions. For example, if one of the cameras is moved by so much as 1cm, the tracking system fails to function. Although automatic recalibration might be possible by using natural or artificial environmental fiducials, in practice these are either difficult to detect or highly intrusive when added to the environment. Thus, cameras being moved or rotated requires retraining the neural network, something a non-technical user would never want to do.

It is not accidental that so much computer vision research around the world is performed in rooms without windows. Computer vision systems, particularly ones that rely on background segmentation, can be extraordinarily sensitive to environmental lighting conditions. For example, early in the Intelligent Room's development, ambient light coming through a newly opened door could completely disrupt all of the room's vision systems. While this was an early design flaw, in general it can be extremely difficult to compensate for changing lighting conditions.

Shadows are also particularly difficult phenomena to cope with and we took great pains to avoid them. We disabled some of the room's overhead fluorescent lighting and used upward pointing halogen floor lamps instead. Additionally, we selected a fairly dark colored carpet, which is better at shadow masking. The tracking system also used a color correction mechanism for shadow elimination. However, a static color correction scheme was only partially useful because the tracking cameras were affected by the dynamic lighting of the projected video displays.

Solutions

Our research agenda for computer vision systems for Hal has changed drastically from the approach used in the Intelligent Room. Rather than incorporating the state of the art in visually based interactions, we have become far more interested in robust vision systems that require little calibration and are essentially self-training.

We have enabled the room's vision systems to reinforce one another. For example, our multi-person tracker may temporarily lose people when they occlude one another or blend into the background. One way to help offset this is to have the finger pointing system provide information useful for tracking. Someone finger pointing at the projected display must be standing somewhere near that position on the room's floor. By knowing where someone is pointing, the tracker can focus its attention on that section of the room. Conversely, the tracking system allows the room to identify the person who is pointing at the wall. By determining which tracked person is closest to the pointed at position, the room can distinguish among its inhabitants during finger pointing gestures.

Various devices in the room can also interact with its

vision systems. The software agents that control the room's drapes and electrical lights notify the vision systems before they do anything that might affect the room's ambient lighting. This allows each vision system either to recalibrate or to deactivate itself until conditions favorable to its correct operation are restored and also avoids incorrect event recognition due to luminosity changes.

Although dynamic person tracking seemed essential during the design of the Intelligent Room, it became clear in retrospect that the vast majority of the tracking system's output is thrown away. Few applications need or can make use of real-time trajectory information for the room's occupants. Rather, what is particularly important is to know where someone is when she stops moving (i.e. next to or sitting on some piece of furniture) or when she has crossed a particular threshold (i.e. the room's doorway).

It is far easier and computationally less demanding to build systems that provide these kind of relatively slowly changing data without resorting to real-time occupant tracking. They look for people at rest in places where they are expected to be found, such as sitting on a couch or standing by a display, or for people crossing through a narrow, well-defined region such as a doorway.

We have implemented and experimented with several such systems, which we call *static person locators* and *threshold detectors*. These include a template-based *couch detector*, which locates people either sitting or lying down on a chair or sofa. This system is easily trained and quite robust. We have also implemented a dedicated *doorway tracker* for distinctly determining when someone enters or leaves the room, and thereby it also keeps track of how many people are currently present. Both of these systems are algorithmically quite simple and far less sensitive to environmental variations than our initial tracking system. They have proved quite robust, and their initial detection accuracy in varying light conditions and over a wide range of individuals is over 90%.

We are also creating generic chair locators using a ceiling mounted vision system. Our assumption is that occlusion of a chair provides evidence that someone is sitting in it, and this person can be located using prior knowledge of the chair's position. This system will use low dimension eigenspaces for approximating object manifolds under varying pose and lighting conditions (Stauffer [21]). The advantage to this approach is that the system need not be given in advance an explicit model of the chairs it will be locating. The system can construct object manifolds itself by having a user rotate any new types of chairs she brings inside.

6. Speech Interactions

Among the earliest decisions regarding the Intelligent Room was that it should support spoken language interactions. In particular, we were interested in allowing

these interactions to be unimodal – i.e. ones that did not tie the user to a video display to verify or correct utterances recognized by the system nor require a keyboard for selecting among possible alternative utterances. We also wanted to avoid simply creating a keyboard replacement that allowed speech commands to substitute for actions ordinarily performed by typing or mouse clicking. Finally, we wanted to allow interaction with multiple applications simultaneously and thus not have interactions that monopolized the user. In the process, we have tried to allow the Intelligent Room to engage in dialogs with users to gather information, correct misunderstandings, and enhance recognition accuracy.

People in the Intelligent Room wear wireless lapel microphones that transmit to the speech understanding system described below. By default, the room ignores the spoken utterances of its inhabitants, which are generally directed to other people within the room. This state is known as “the room being asleep.”¹ To obtain the room's attention, a user stops speaking for a moment and then says the word “*Computer*” out loud. The room immediately responds with an audible, quiet chirp from an overhead speaker to indicate it is paying attention. The user then has a two second window in which to begin speaking to the room. If the room is unable to recognize any utterances starting within that period, it silently goes back to sleep until explicitly addressed again. However, if what the user says is recognized, the room responds with an audible click and then under most circumstances it returns to sleep. This hands- and eyes-free style of interaction coupled with audio feedback allows a user to ignore the room's *computational presence* until she explicitly needs to communicate with it. There is no need to do anything other than preface spoken utterances with the cue *Computer* to enable verbal interaction. Thus, a user can interact with the room easily, regardless of her proximity to a keyboard or monitor.

The Intelligent Room is capable of addressing users via the Festival Speech Synthesis System (Black et al. [2]). Utterances spoken by the room are also displayed on a scrollable LCD sign in case a user was unable to understand what was said. The room uses its speech capability for a variety of purposes that include conducting dialogs with users and getting its occupant's attention without resorting to use of a visual display. Sometimes, the room chooses to respond vocally to a question because its video displays are occupied by what it considers high priority information. For example, if a user asks, “*What's the weather forecast for New York City?*” the room can simply read the forecast to the user, rather than put up a weather map containing forecast information if its displays are occupied.

For processing spoken utterances, we use both the Summit (Zue et al. [27]) and DragonDictate speech recognition systems in parallel. Each of these has

¹ The room's vision systems continue to function and respond to users even when it is not listening for verbal input.

different strengths and used together they have fairly robust performance. The Summit system recognizes continuous speech and is particularly adept at handling syntactic variability during recognition. By entering bigram models, it is fairly straightforward to build topically narrow but syntactically unconstrained sets of recognizable utterances. Bigram models, however, make it quite difficult to exclude particular statements from being erroneously recognized by the system and require that we heavily post process Summit's output. This is performed primarily by the START natural-language information retrieval system (Katz [15]).

DragonDictate is a commercially available system primarily used for discrete speech dictation, meaning that users must pause after each word. This, when coupled with its relatively low word accuracy, would be an intolerable speech interface to the room. However, DragonDictate also supports explicit construction of continuous speech, context-free recognition grammars. Via a special library, it also provides complete control over low-level aspects of its behavior to external applications, which makes it ideal for incorporating into other systems.

Issues

There is a tradeoff between making the room's recognition grammars sufficiently large so that people can express themselves somewhat freely versus making the grammars small enough so that the system runs with high accuracy and in real-time. We tuned DragonDictate's performance by creating sets of specialized grammars for different room contexts and having the room's software agent controller dynamically activate different subsets of grammars depending on the context of the activity in the Intelligent Room (Coen et al. [9]). This allows us to overcome the combinatorial increase in parsing time due to incorporating natural syntactic variability in the recognition grammars.

Instead of keeping a single enormous recognition grammar active, the room keeps subsets of small grammars active in parallel, given what it currently expects to hear. The key assumptions here are that certain types of utterances are only likely to be said under particular circumstances, and these are circumstances among which the room is capable of distinguishing. These may be related to where someone is spatially, the history of her previous interactions (i.e. what room applications are active), how she is gesturing, what devices in the room are doing, etc. At the simplest level, this can range from the implausibility of someone saying "*stop the video*," when none is playing, to more complex dependencies, such as the meaninglessness of someone asking "*What's the weather there?*" if no geographic entity has somehow been brought to the room's attention.

We have generalized the notion of linguistic context to include the state of and goings on in the room and have put this contextual knowledge into the room's software agents rather than its linguistic data structures. For

example, if the room starts showing a video clip, the agent that controls the showing of videos activates the grammars that involve VCR operation. When the clip stops, these grammars are in turn deactivated. More interesting cues can involve the location of someone inside the room. The fact that someone has moved near an interactive displayed map causes the room to pay attention to spoken utterances involving geographic information. Thus, information from the room's other systems can help overcome computational limitations in the room's speech recognition and understanding systems.

Verbal interactions can also be extremely useful for dealing with the room's other modalities. They can be used to gather information about what the room is observing, to modify internal representations of its state, or to correct a perceptual error. It is also of enormous benefit to be able to verbally interact with the room's vision systems while developing or debugging them, because it is generally impossible to manually interact with them at a workstation while remaining in the cameras' fields of view.

7. Conclusion

Our experience with the Intelligent Room has led us to reevaluate many of our initial assumptions about how a highly interactive environment should be designed. Intelligent environments need to be more than rough assemblages of previously existing systems. In particular, careful selection and communication among modalities can lead to synergistic reinforcement and overall, a more reliable system. The modalities must also be carefully selected in order to make the environment easy to install, maintain, and use under a wide range of environmental conditions.

Systems that dynamically adjust to the room's activity, such as our speech understanding system, and systems that can train themselves and avoid extensive manual calibration, are essential to an IE's success. We hope the issues addressed in this paper will both stimulate further discussion and prove useful to other designers of intelligent environments.

8. Acknowledgements

Development of the Intelligent Room has involved the efforts of many people. This includes Professors Tomas Lozano-Perez, Rodney Brooks, and Lynn Stein. Graduate students who have been or are involved in the room include Mark Torrance, Jeremy De Bonet, Chris Stauffer, Sajit Rao, Darren Phi Bang Dang, JP Mellor, Gideon Stein, Michael Coen, Josh Kramer, Brenton Phillips, Mike Wessler, and Luke Weisman. Postdocs associated with the project include Kazuyoshi Inoue and Polly Pook. Many undergraduates have been or are currently working on it. These include Kavita Thomas, Nimrod Warshawsky, Owen Ozier, Marion Groh, Joanna Yun,

James Clark, Victor Su, Sidney Chang, Hau Hwang, Jeremy Lilley, Dan McGuire, Shishir Mehrotra, Peter Ree, and Alice Yang.

References

1. Abowd, G., Atkeson, C., Feinstein, A., Hmelo, C., Kooper, R., Long, S., Sawhney, N., and Tani, M. Teach and Learning a Multimedia Authoring: The Classroom 2000 project. *Proceedings of the ACM Multimedia'96 Conference*. 1996.
2. Black, A. and Taylor, P. Festival Speech Synthesis System: system documentation (1.1.1) Human Communication Research Centre Technical Report HCRC/TR-83. University of Edinburgh. 1997.
3. Bobick, A.; Intille, S.; Davis, J.; Baird, F.; Pinhanez, C.; Campbell, L.; Ivanov, Y.; Schütte, A.; and Wilson, A. Design Decisions for Interactive Environments: Evaluating the KidsRoom. *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
4. Cohen, P., "The role of natural language in a multimodal interface," *Proceedings of User Interface Software Technology (UIST'92) Conference*, Academic Press, Monterey, California, 1992.
5. Cohen, P., Chen, L., Clow, J., Johnston, M., McGee, D., Pittman, K., and Smith, I. Quickset: A multimodal interface for distributed interactive simulation, *Proceedings of the UIST'96 Demonstration Session*, Seattle. 1996.
6. Coen, M. Building Brains for Rooms: Designing Distributed Software Agents. *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*. (IAAI'97). Providence, R.I. 1997. <http://www.ai.mit.edu/people/mhcoen/brain.ps>
7. Coen, M. Towards Interactive Environments: The Intelligent Room (a short paper). *Proceedings of the 1997 Conference on Human Computer Interaction (HCI'97)*. Bristol, U.K. 1997.
8. Coen, M. (ed.) *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
9. Coen, M.; Thomas, K.; Weisman, L.; Groh, M.; and Yee, A. A Natural Language Modality for an Embedded Multimodal Environment. Forthcoming.
10. Cooperstock, J.; Fels, S.; Buxton, W. and Smith, K. Environments: Throwing Away Your Keyboard and Mouse. *Communications of the ACM*. 1997.
11. Dang, D. Template Based Gesture Recognition. SM Thesis. Massachusetts Institute of Technology. 1996.
12. Davis, J. and Bobick, A. The representation and recognition of action using temporal templates. *Proceedings Computer Vision and Pattern Recognition (CVPR'97)*. pp.928-934. 1997.
13. DeBonet, J. Multiple Room Occupant Location and Identification. 1996. http://www.ai.mit.edu/people/jsd/jsd.doit/Research/HCI/Tracking_public
14. Druin, A.; and Perlin, K. Immersive Environments: a physical approach to the computer interface. *Proceedings of the Conference on Human Factors in Computer Systems (CHI'94)*, pages 325-326. 1994.
15. Katz, B. Using English for Indexing and Retrieving. In *Artificial Intelligence at MIT: Expanding Frontiers*. Winston, P.; and Shellard, S. (editors). MIT Press, Cambridge, MA. Volume 1. 1990.
16. Lien, J., Zlochow, A., Cohn, J., Li, C., and Kanade, T. Automatically Recognizing Facial Expressions in the Spatio-Temporal Domain. *Proceedings of the Workshop on Perceptual User Interfaces (PUI'97)*. Alberta, Canada. pp.94-97. 1997.
17. Lucente, M.; Zwart, G.; George, A. Visualization Space: A Testbed for Deviceless Multimodal User Interface. *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
18. Mozer, M. The Neural Network House: An Environment that Adapts to its Inhabitants. *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
19. Newman, W. and Wellner, P. A Desk Supporting Computer-based interaction with paper. *Proceedings of the Conference on Human Factors in Computing Systems (CHI'92)*. p587-592. 1992.
20. Saund, E. Example Line Drawing Analysis for the ZombieBoard Diagrammatic User Interface. <http://www.parc.xerox.com/spl/members/saund/lda-example/lda-example.html>. 1996.
21. Stauffer, C. Adaptive Manifolds for Object Classification. 1996. <http://www.ai.mit.edu/people/stauffer/Projects/Manifold/>
22. Stiefelhagen, R., Yang, J., and Waibel, A. Tracking Eyes and Monitoring Eye Gaze. *Proceedings of the Workshop on Perceptual User Interfaces (PUI'97)*. Alberta, Canada. pp.98-100. 1997.
23. Torrance, M. Advances in Human-Computer Interaction: The Intelligent Room. *Working Notes of the CHI 95 Research Symposium*, May 6-7, Denver, Colorado. 1995.
24. Want, R.; Schilit, B.; Adams, N.; Gold, R.; Petersen, K.; Goldberg, D.; Ellis, J.; and Weiser, M. The ParcTab Ubiquitous Computing Experiment. Xerox Parc technical report.
25. Weiser, M. The Computer for the 21st Century. *Scientific American*. pp.94-100, September, 1991.
26. Wellner, P. The DigitalDesk Calculator: Tangible Manipulation on a Desk Top Display, *Proceedings of UIST'91*. pp.27-33. 1991.
27. Zue, V. Human Computer Interactions Using Language Based Technology. *IEEE International Symposium on Speech, Image Processing & Neural Networks*. Hong Kong. 1994

LEARNING SPATIAL EVENT MODELS FROM MULTIPLE-CAMERA PERSPECTIVES

Michael H. Coen and Kevin W. Wilson
Human Computer Interaction Group
MIT Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139
{mhcoen, kwilson}@ai.mit.edu

Abstract. Intelligent, interactive environments promise to drastically change our everyday lives by connecting computation to the ordinary, human-level events happening in the real world. This paper describes a new model for tracking people in a room through a multi-camera vision system that learns to combine event predictions from multiple video streams. The system is intended to locate and track people in the room, determine their postures, and obtain images of their faces and upper bodies suitable for use during teleconferencing. This paper describes the design and architecture of the vision system and its use in Hal, the most recently constructed interactive space in our Intelligent Room project.

1 Introduction

For the past four years, the MIT Artificial Intelligence (AI) Laboratory has been developing platforms for research in Human-Computer Interaction (HCI) as part of the Intelligent Room Project. This work has been motivated by the following simple observation: computers today are generally used for things that are computational, such as reading e-mail, and for most of us, the majority of our lives are spent doing non-computational things, such as taking baths and eating dinner. Historically, however, computational systems have had no connection with the ordinary, human-level events going on in the world around them. Thus, they have no way to participate in the everyday lives of the people who use them.

In the Intelligent Room Project, we are interested in creating spaces – generally known as Intelligent Environments (IEs) – in which computation is seamlessly used to enhance ordinary, everyday activities. We want to incorporate computers into the real world by

embedding them in regular environments, such as homes and offices, and allow people to interact with them the way they do with other people. The user interfaces of these systems are not menus, mice, and keyboards, but instead gesture, speech, affect, context, and movement. Their applications are not word processors and spreadsheets, but smart homes and personal assistants. Instead of making computer interfaces for people, we believe it is of more fundamental value to make people interfaces for computers.

Fortunately, over the past few years, developments in the field of computer vision have begun to provide many opportunities for exploring new types of HCI. These allow computational systems to, among other things, determine people's identity, physical location, gaze direction, facial expression, hand gestures, and activities (for example, see [8,13]). However, the requirements placed on machine vision systems used for IEs are significantly different from the requirements placed on machine visions systems used in laboratory or industrial settings. A vision system that works beautifully under structured, carefully orchestrated laboratory conditions may be of little use in environment where people are encouraged to interact naturally, without worrying about which direction they are facing or even whether they stay in the field of view of some camera.

This paper describes a new model for tracking people in an IE in the context of *Hal*, our most recently constructed intelligent room. In this model, locations in a room that are likely to contain people are used as reference points to interactively train a multi-camera vision system. It learns to combine event predictions from multiple video streams in order to locate people at these reference points in the future. The system can also dynamically track people as they move between these locations. Because most rooms have natural attractors for human activity, such as doorways, furniture, and displays, the selection of training points is generally readily apparent from the layout of the room.

Our interest is how to make it easy to set up and maintain this type of vision system in an environment such as Hal, which currently contains four fixed and three

This material is based upon work supported by the Advanced Research Projects Agency of the Department of Defense under contract number F30602-94-C-0204, monitored through Rome Laboratory.

computer-steerable cameras. The vision system described here is intended to locate and track people in the room, determine their postures, and obtain images of their faces and upper bodies suitable for teleconferencing.

Although it may seem extravagant to outfit a small room with so many cameras, the increasingly low-cost of hardware for machine vision makes it quite feasible to do so. However, installing a large number of vision systems in a single room raises the following questions, which the rest of this paper addresses:

1. How can the vision systems learn the types of events they can see?
2. How can individual cameras be shared by multiple vision systems?
3. How can we learn the correspondences among events in the fields of view of different cameras?
4. How can the vision systems learn how their visual fields overlap so they can reinforce each other through communication?
5. How can we avoid the need to precisely calibrate the cameras?

We argue in this paper for a particular approach that has been valuable in the development of Hal, which has been equipped with computer vision systems specifically designed for coping with the difficulties expected to arise during unstructured interactions. The primary components of this system's architecture are its generalizability, redundancy, and ability to learn spatial events without requiring elaborate training scenarios.

We first discuss design level issues for the system, including why we used computer vision as opposed to other possible technologies. Next, we describe Hal's layout, components, and the types of applications intended to run within it. Finally, we discuss the architecture of the vision systems and how well this system has satisfied our design goals.

2 Design Considerations

Research in Intelligent Environments has been heavily influenced by the work done in Xerox PARC's Ubiquitous Computing (UbiComp) Project [14,15]. UbiComp has avoided AI approaches to HCI for many of the perceptory tasks at which computer vision excels. Instead, hardware sensors – more reliable in the early 1990s but highly limited in their perceptory capabilities –

were employed to gather information about people interacting with UbiComp systems. For example, someone could be tracked in a building by wearing an infrared-transmitting active badge. Similarly, a room could determine that a person was sitting in a chair via a pressure switch in the seat cushion.

However, machine vision techniques can do a far better job of allowing IEs to understand the activities going on within them than can simple sensing technologies. For example, although a pressure sensor in a chair may be able to register that someone has sat down, it is unlikely to provide any other information about her, such as who she is or which way she is looking; active badges or motion detectors [9] can indicate that someone has entered a room, but they can not, for example, determine what objects she is pointing at while speaking. Computer vision systems can provide much richer descriptions of human activities, which are essential for supporting natural interactions.

Non-vision technologies are also highly encumbered – furniture must be constructed or retrofitted with sensors and people must clip devices onto their bodies for a UbiComp system to be aware of them. Vision systems sense at a distance, so that people and furniture do not need to any special augmentation for an IE to perceive them. They also have far greater flexibility, in that once the video hardware is installed, different image processing algorithms can be applied to the video streams without requiring changes to the system's physical infrastructure. Finally, their video streams are reusable by other systems, such as in teleconferencing or surveillance applications.

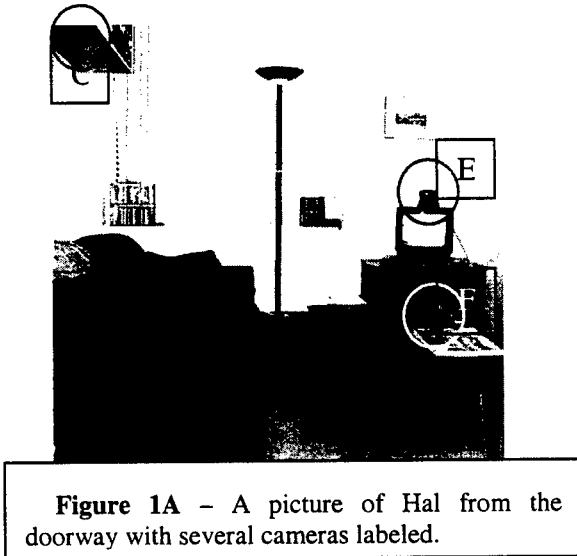
2.1 Vision for IEs

Using machine vision systems for IEs, as opposed to for more general-purpose HCI applications, can change basic assumptions about what types of data the systems are providing. IEs frequently do not require high-precision information, but they require that information be supplied on a time scale comparable to the events they are observing. In addition, people are comfortable interacting in very complex, unstructured environments, so it is reasonable to require that the vision systems be capable of functioning within them as well.

Much of the information needed in human-computer interaction is qualitative in nature. For example, it may be necessary to distinguish between a person sitting on a

chair, a person standing in front of the chair, and a person standing by the door. This information fits more accurately into a qualitative classification system than into a system that gives only quantitative information (such as position and size).

Since much useful information can be described qualitatively and since qualitative information should be obtainable without a precise geometric model of the world, we insist that precise external calibration be avoided in our vision systems. By minimizing the amount of required external calibration, we expect it



should be easier both to modify the configuration of the system and to make it robust with respect to small perturbations, such as a camera being moved accidentally. We view both of these characteristics as essential to making intelligent environments viable in the “real-world” – outside of controlled laboratory conditions.

Information from machine vision systems used in a noisy, unconstrained environment often possesses a large degree of uncertainty. Some sources of the uncertainty include unfavorable lighting conditions and coincidental color matches between objects – for example, a person wearing a green shirt who happens to be sitting in a green chair may seem to disappear to many vision systems. One way to reduce the amount of uncertainty in the system is to replicate the systems and place additional cameras at different locations in the

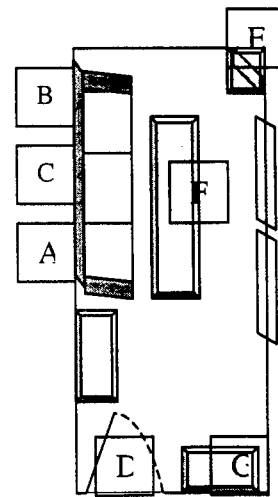


Figure 1B – Hal's floor plan with all camera positions labeled.

environment. Confusing shadows that are visible for one camera may not be problematic for a second camera viewing the scene from a different perspective, and the overlap between foreground and background objects will likely be different for them as well.

3 Interacting with Hal

Hal is a small room within our lab that doubles as the first author's office (See Figures 1A and 1B). Hal is equipped with microphones, seven video cameras, and a variety of audio-visual output devices that it can directly control. The cameras, microphones, and devices are connected to a cluster of a dozen workstations in an adjacent room. These workstations run a distributed software agent system that controls Hal's operation [2].

Hal was designed to explore a wide range of interactions involving futuristic residential spaces – stressing quality of life – and commercial spaces – stressing information management. We have therefore created applications in Hal that support entertainment, teleconferencing, business meetings, military command post scenarios, and information retrieval. (More in-depth descriptions of these applications can be found in [3,5].) Hal was carefully designed with respect to its intended applications [3]. Because we were particularly interested in creating teleconferencing and meeting recording

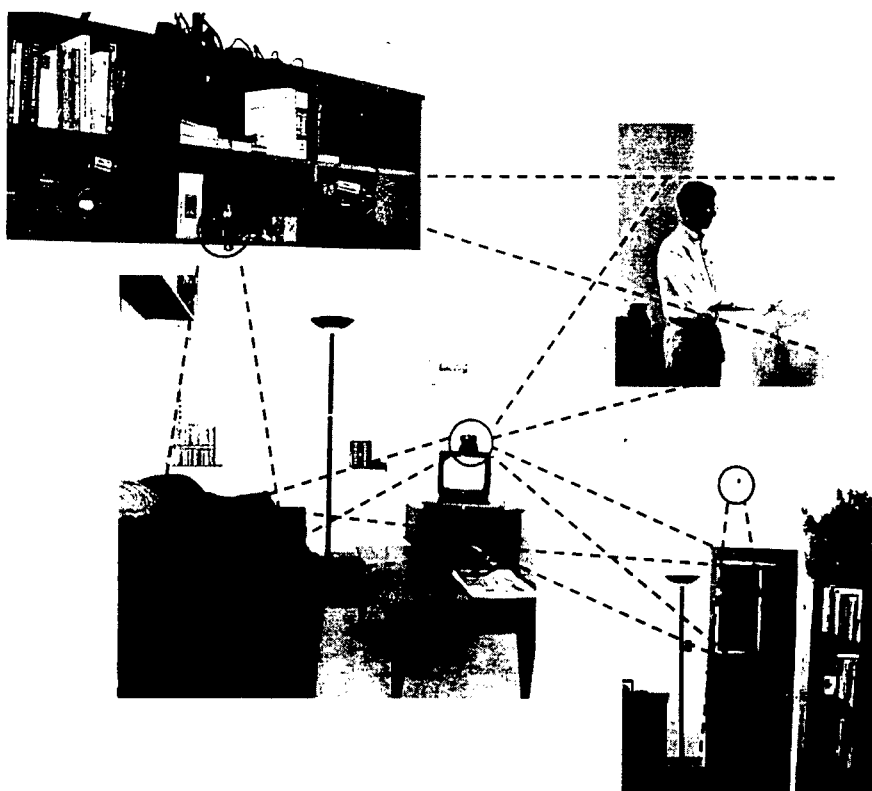


Figure 2 – Hal's layout. Cameras are circled in the image and individually labeled. Sections of each camera's field of view are delineated with dotted lines.

systems within Hal, cameras had to be strategically placed to obtain useful views of Hal's inhabitants. Many visually based room tracking systems, such as [1], use fixed, ceiling-mounted cameras in very large rooms, because they can obtain "bird's eye" views and are less sensitive to visual occlusion. However, cameras placed in or near a ceiling cannot typically obtain images of people that are suitable for other people to watch. It was therefore necessary to place cameras inside Hal that were able to obtain high-quality images of its inhabitants' faces.

We also wanted these cameras to track people during teleconferencing, so that users could freely move about the room; people who are teleconferencing should not need to concern themselves with staying in the field of view of a particular camera. However, the proximity of the users and teleconferencing cameras made it likely that people might quickly move out of an individual camera's field of view. We therefore opted for hybrid approach, combining fixed "bird's eye" cameras outfitted with wide-angle lenses – that could see large sections of Hal – with steerable camera that could quickly focus on much smaller sections of the room. Our goal has been to allow these systems to intercommunicate to overcome their inherent individual limitations.

Of Hal's seven cameras, two are dedicated to a special purpose vision system that allows a user to manipulate a computer's mouse cursor by pointing a laser pointer at either of two projected video displays. These are labeled A and B in the figure, and will not be discussed further in this paper. Cameras C & D are fixed video cameras with wide-angle lenses. Camera C is mounted in an overhead bookshelf and views the couch and coffee table areas. Camera D is ceiling mounted and overlooks the doorway. Cameras E, F, and G are Sony EVI-D30 steerable cameras under computer control. Each of these cameras has a pan range of approximately 180 degrees and tilt range of approximately 90 degrees. Figure 2 contains several views of Hal and illustrates how the fields of view of the cameras overlap.

The following scenario is an amalgam of several illustrative interactions that currently run inside Hal. (Capital letters in the following refer to cameras, e.g., "E" is Camera E in the floor plan.) The first author walks inside Hal. Hal sees him via D and rotates whichever of E or F is not otherwise occupied to obtain an image of his face. Hal can currently detect the presence

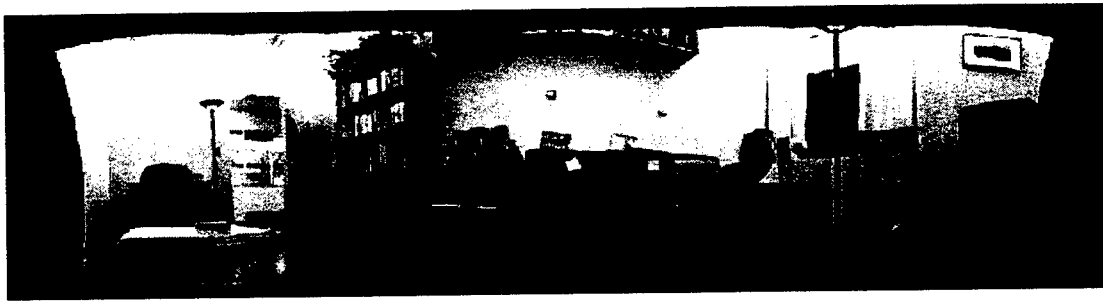


Figure 3 – Sample panoramic background generated by Camera F from 18 source images and stored in transformed pan-tilt space. Pan range = 180° and tilt range = 60°

of faces in images but cannot yet distinguish people by them. However, seeing a face in the doorway increases the confidence that someone actually just entered the room. Also, this image can be videotaped or transmitted during a teleconferencing session so that remote participants can see who has entered. Hal says, "Good evening Michael, you have three messages waiting." Hal displays the messages on one of the projected displays. As the user walks to the couch and sits down, Hal will attempt to track him via E or F. Once the user is seated on the sofa, Hal sees him via C and keeps F focused on his upper torso. As the user moves around the sofa/coffee table area, Hal tracks him with F. C is used to help steer F in case the user moves out of F's field of view. If the user moves too close to F and Hal can no longer obtain a good image of him, Hal will switch to E (or eventually perhaps G, which due to temporary hardware limitations, does not currently interoperate with E). Next, the user lies down on the sofa. Supposing his clothing is quite dark, he blends into the background so Hal isn't sure if he has actually lay down. Hal will then search for his face via C and will sweep F to make sure it cannot see him sitting up. Hal will then infer that he did indeed lie down; it will next ask if he is going to relax for a little while. If the user answers affirmatively, Hal will dim the lights, close the drapes, and put on Mozart softly in the background. After Hal dims the lights, it will update its model of what the room looks like.

4 Implementation

The current implementation of Hal's vision systems contains three major parts. At the lowest level is a common image processing library that processes the video streams from each of Hal's cameras (with the exception of A and B, as mentioned above). In programmer's parlance, the same "code" is running over all the video streams. The data obtained from each stream, in the form of a feature vector, is then passed up to a higher-level system of Image Processing Agents that build statistical cluster-based models to uniquely classify the events seen from each camera's viewpoint. Recursively, these single-camera events are then themselves temporally clustered, creating a global qualitative model of how a single perceptual event within Hal is simultaneously perceived by any of the cameras that can see it. For example, when Hal builds a model of someone walking through the doorway, this model contains what it expects Cameras D, E, and F would see if they were watching someone do so.

These multi-viewpoint models can then used to increase confidence that events are being detected correctly. For example, consider the case where Hal thinks someone is entering the room because of data obtained by Camera D. If Hal knows that this event corresponds to a certain (*pan*, *tilt*) value for Camera E, Hal can "borrow" this camera for a moment (presuming it is being used for something else) and turn it to that

orientation to supplement that data obtained from Camera D.

We now discuss each of the vision system's levels in turn.

4.1 Low-level image processing

Each camera is connected to a separate workstation equipped with a frame grabber and running a low-level image-processing library. This library transforms each incoming image into a feature vector containing background difference, skin detection, and face detection information. (Optical flow values can also be calculated for each video stream, but this is currently only done for Camera D.) These vectors are then available for the Image Processing Agents described in the next section. The low-level libraries process images somewhere between 10-30Hz, depending on what they are viewing. For example, an image containing a face takes longer to process than an image without one.

We discuss here the three primary components of the feature vector and the optical flow calculation.

Background Differencing

Although background differencing itself is a relatively simple technique, the fact that the system uses steerable cameras adds significantly to the complexity of the process. Although it is straightforward to create an array of background images for a small number of predetermined (*pan*, *tilt*) values for a steerable camera, it is not feasible to capture and store a background image for each of the thousands of possible pan-tilt orientations.

It is possible, however, to completely span the camera's viewable world using only a few dozen images. By then performing appropriate coordinate transforms, it becomes feasible to dynamically synthesize a background for an arbitrary camera orientation. We currently use eighteen images (two rows by nine columns of slightly overlapping images) to span each camera's range of 180 degrees of pan and 60 degrees of tilt. The system uses these images to create one large "panoramic" background (Figure 3), which is stored in the transformed pan-tilt space instead of x-y coordinates of a certain image position.

A myriad of issues must be dealt with while building panoramic backgrounds. This includes selecting an appropriate scaling for the background that avoids undersampling the images. The coordinate transform from camera image x-y space to panoramic background pan-tilt

space also depends on several parameters associated with the camera. Space limitations preclude detailed discussion, but these include the camera's aspect ratio, focal length, radial distortion, and precise axis of rotation. For example, the cameras used by the system are not ideal; they exhibit significant but consistent radial distortion due to imperfections in the lenses. This type of distortion can be modeled by determining a "center of distortion" and by radially expanding or contracting the image according to an empirically determined power series. Although it was decided that empiric calibration would be most expedient, at some point it may be easier to automate much of the calibration using the procedure described in [12].

Skin Color Detection

The second algorithm used by the low-level system is a skin-color detection algorithm that operates independently on each input pixel. Because skin color is determined by a combination of the red hue in blood and the yellow-brown hue of melanin, the hue of skin is restricted to a narrow range of hues. To detect input pixels in this narrow range of hues, the input RGB values were transformed into a log-opponent color parameterization according to a variant of the method described in [6].

Face Detection

The third algorithm that is applied to the input is a face detection algorithm. The algorithm is a ratio template algorithm developed by [11] and used in the Cog humanoid robot [10]. This algorithm is quite computationally efficient when compared to neural-network approaches, and it performs reasonably well in a cluttered environment. These features make it suitable for human-computer interaction scenarios.

Optical Flow

A correlation-based optical flow algorithm compares localized patches of an image to patches of a subsequent image with a range of possible offsets from the original patch. The best match in the subsequent image is assumed to represent the location of the corresponding patch in the new image, and the spatial offset between patches in the two images corresponds to the optical flow. Optical flow information is assumed to be the same as the motion field of the image. This information is therefore useful for detecting transient events such as entries and exits. These events happen too quickly to be reliably detected by the other low-level image processing components.

4.2 Local Event Processing – Image Processing Agents

The next level of the system is comprised of *Image Processing Agents*, which uniquely connect with the different low-level libraries processing the incoming video streams. Each Image Processing Agent receives a stream of feature vectors corresponding to the visual input of the camera with which it is connected. In order to reduce the communication bandwidth between the low-level systems and the Image Processing Agents, which generally all run on different workstations, background difference and skin color information are not transmitted for each pixel. Instead, the image is divided into a 10x10 cell grid, and average values are reported for each cell. Currently, up to two faces can be reported for each image, where a face notification consists of its size and location in pan-tilt space. (The limit of two faces was arbitrarily chosen to simplify the communication protocol and can easily be increased.)

Each Image Processing Agent therefore receives a 209 dimensional feature vector, consisting of 100 background difference values, 100 skin-tone match values, 6 faces values, the center of the image in pan-tilt coordinates, and a measure of the overall brightness of the image, at a frequency of somewhere between 10-30Hz.

The agent uses these vectors to build statistical models of the events its camera is capable of detecting during a training session described in the next section. After building these models, it attempts to classify future incoming data to determine if any of the previously learned events reoccur. When they do, this information is passed to the global event processing system described in the next section. The agent is also responsible for several other tasks. The first of these is maintaining the quality of its background image, against which differences are computed to perform foreground object segmentation. Persistently unrecognized phenomena, such as a region being occluded without skin-tones being present, can lead the agent to incorporate that region into the

background model. Agents that are connected to steerable cameras can also steer the cameras to track moving objects that catch their attention, even if these objects move away from the location of a previously learned event. The dynamic background synthesis performed by the low-level library makes this behavior possible.

The Image Processing Agent constructs a window W of approximately two seconds worth of the most recently received feature vectors. We will call the size of this window k , so $W = [\bar{v}^i \dots \bar{v}^{i+k}]$. It uses this window to construct and subsequently update two special vectors after each new feature vector arrives. The first is an mean vector $\bar{\bar{v}}_j = \text{mean}(\bar{v}_j^i, \dots, \bar{v}_j^{i+k})$. The other is defined over the standard deviations of the dimensions in the feature vectors, $\bar{\sigma}_j = \sigma(\bar{v}_j^i, \dots, \bar{v}_j^{i+k})$. Event learning and classification happens over these vectors, rather than over the incoming feature vectors directly.

We define a *local event* E as a tuple $(\bar{\bar{v}}^i, \bar{\sigma}^i)$ of vectors constructed from a window *observing* the event. By this, we mean that the physical event corresponding to E was occurring in the camera's foveal area while the window was being constructed. Each agent builds up a set of local events, LE , that it is capable of distinguishing during a training process discussed in the next session. (Each agent also constructs at least one "null" event.) Once the agent has been trained, it constantly attempts to classify the event "contained" in the window as one of its previously learned events.

To determine the similarity between a currently observed window and a previously observed event, we define the *distance* between a window W (with $\bar{\bar{v}}$) and a local event $E_j = (\bar{\bar{v}}^j, \bar{\sigma}^j) \in LE$, letting $n = |\bar{\bar{v}}|$, as:

$$\text{distance}(W, E_j) = \frac{1}{\sqrt{n}} \sum_{i=1}^n \frac{|\bar{\bar{v}}_i - \bar{\bar{v}}_i^j|}{\bar{\sigma}_i^j + \epsilon}$$

This function very much resembles a renormalization according to a t-distribution with zero mean and unit variance. However, that is not our intended use for it. Rather, we are using it as a means to calculate a weighted

distance between what has currently been observed in the window W and some previously observed event E_j , where dimensions are inversely weighted in proportion to how noisy their signals are. The ϵ factor is used to avoid difficulties in case a particular dimension has zero variance.

The agent selects $\min_{E_j \in LE} (\text{distance}(W, E_j))$ as the most likely event to be occurring. However, events can be partially activated, in that multiple explanations for whatever is currently being observed will be passed on to the global event system discussed in the next session. It is expected in this case that data from other cameras will help disambiguate the room-level event actually being observed. If an agent is asked to verify whether a local-event is occurring as part of this determination, it will attempt to sense the event by forcing whichever feature-vector dimensions it has control over to take on the values associated with the event in question. At present, this is limited to pan-tilt values corresponding to the local event; asserting these values causes the low-level library to move the camera to that orientation.

4.3 Global Event Processing

Global events correspond to qualitative models of how a single perceptual event within Hal is simultaneously perceived by all of the cameras that can see it. There are two mechanisms through which global events are created and/or updated. The first is through an explicit training scenario in which Hal guides users through a series of interactions. The second is a more complicated mechanism where Hal searches for statistically significant temporal co-occurrences of events and infers they are related.

Hal is built on top of a subset of a Room-Perception Application Programming Interface (API) that describes the types of events an abstract Intelligent Environment might sense going on within it, which we simply call *room events*. People writing applications for Hal interface with the perceptual systems via this API, so they can, for example, write code that refers to someone sitting on the middle of couch without having any knowledge of how the perceptual systems work or how this event is recognized by the room. The subset of the API Hal implements has a hierarchy of events within it, such as "Room Enter," "Room Exit," "Sitting Down," "Sitting on Couch," "Standing by Object," etc.

When Hal is initially configured, it leads the user through a training scenario in order to learn what it is like to observe these events. For each fixed camera, this is simply a matter of determining whether there is an unrecognized event in its field of view. The steerable cameras, however, are swept through their range of pan-tilt values in order to locate a new event. Once they have found and centered it, they construct a local event model as described above to represent it. For example, training the "Room Enter" event occurs as follows. Hal says, *"Hello, please enter the room."* The user then walks inside and stands in the doorway. Hal finds them via Camera D and says, *"Please remain still for one moment."* Hal then sweeps the steerable cameras looking for an unrecognized event. As the cameras find and center the user, Hal says *"Aha! I've found you! Thank you for training."* A more sophisticated approach might involve the vision systems exchanging color histogram information to insure they are indeed looking at the same event. However, we assume that simultaneous new room events do not occur during training, so there is currently no need to resolve them.

By determining which local events are constructed and/or activated while training a room-level event. Hal builds qualitative models of the room event with reference to them. If the user does something which triggers the associated local-events of a room event, Hal triggers the appropriate Room-Perception API event, such as "Standing by bookcase," and sends it to any applications that have registered they are interested in such events.

A natural extension of the system at this level would be to build a Markov model of transitions among events from observations of typical usage patterns. This model could be used during event transitions to turn steerable cameras preferentially towards events that are more likely to occur, before they actually happen.

Room events can also be partially activated, meaning some of their composite events have been triggered but not with sufficient confidence to actually be sure the event has taken place. Hal will then try to assert the non-active local events according to the mechanism described above to see if they are currently taking place. Simple statistical confidence intervals over these local events are then used to determine whether something of interest has actually happened. Some care must be taken with this mechanism to insure that events that are transient, such as someone walking through a doorway, have not actually completed by the time the steerable cameras turn to check for them. For this reason, the overhead camera in the doorway produces only optical flow information at 30Hz, so that its local-events are activated as quickly as possible. Within less than .5 seconds of someone walking through the doorway, whichever steerable camera is free to verify the event is already turning to look at it.

We are also interested in using this mechanism to connect the room's visual perception with its other

sensory modalities. For example, we have empirically determined that people tend to speak about objects they are closer to and therefore we dynamically bias Hal's speech recognition models with information obtained from its tracking system. These biases are currently hand-coded, but it seems feasible to learn them via temporal correspondences in a manner very similar to how Hal's visual event acquisition currently operates.

References

1. Bobick, A.; Intille, S.; Davis, J.; Baird, F.; Pinhanez, C.; Campbell, L.; Ivanov, Y.; Schütte, A.; and Wilson, A. Design Decisions for Interactive Environments: Evaluating the KidsRoom. *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
2. Coen, M. Building Brains for Rooms: Designing Distributed Software Agents. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*. (IAAI97). Providence, R.I. 1997.
3. Coen, M. Design Principles for Intelligent Environments. In *Proceedings of The Fifteenth National Conference on Artificial Intelligence*. (AAAI98). Madison, Wisconsin. 1998.
4. Coen, M. (ed.) *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
5. Coen, M. The Future Of Human-Computer Interaction or How I learned to stop worrying and love My Intelligent Room. *IEEE Intelligent Systems*. March/April. 1999.
6. Forsyth, D. and Fleck, M. Finding Naked People. *European Conference on Computer Vision*, Volume II, pp. 592-602. 1996.
7. Lien, J., Zlochow, A., Cohn, J., Li, C., and Kanade, T. Automatically Recognizing Facial Expressions in the Spatio-Temporal Domain. *Proceedings of the Workshop on Perceptual User Interfaces (PUI'97)*. Alberta, Canada. pp.94-97. 1997.
8. Lucente, M.; Zwart, G.; George, A. Visualization Space: A Testbed for Deviceless Multimodal User Interface. *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
9. Mozer, M. The Neural Network House: An Environment that Adapts to its Inhabitants. *Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
10. Scassellati, B. Eye Finding via Face Detection for a Foveated, Active Vision System, *Proceedings of the National Conference on Artificial Intelligence*, Madison, WI. 1998.
11. Sinha, P. Object Recognition via image invariants: A case study, *Investigative Ophthalmology and Visual Science*, 35, pp. 1735-1740. 1994.
12. Stein, G. Internal Camera Calibration using Rotation and Geometric Shapes, Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA. 1993.
13. Stiefelhagen, R., Yang, J., and Waibel, A. Tracking Eyes and Monitoring Eye Gaze. *Proceedings of the Workshop on Perceptual User Interfaces (PUI'97)*. Alberta, Canada. pp.98-100. 1997.
14. Want, R.; Schilit, B.; Adams, N.; Gold, R.; Petersen, K.; Goldberg, D.; Ellis, J.; and Weiser, M. The ParcTab Ubiquitous Computing Experiment. Xerox Parc technical report.
15. Weiser, M. The Computer for the 21st Century. *Scientific American*. pp.94-100, September, 1991.

A Context Sensitive Natural Language Modality for an Intelligent Room

Michael Coen, Luke Weisman, Kavita Thomas, and Marion Groh

MIT Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139
{mhcoen, luke, marion}@ai.mit.edu, kavita@cs.cmu.edu

Abstract. This paper describes the design and implementation of a natural language interface to a highly interactive space known as the Intelligent Room. We introduce a data structure called a *recognition forest*, which simplifies incorporation of non-linguistic contextual information about the human-level events going on in the Intelligent Room into its speech understanding system. This aim of using context has been to allow multiple applications—all of which support a relatively high degree of natural syntactic variability—to run simultaneously in the Intelligent Room.

1 Introduction

This paper describes the design and implementation of the natural language interface to the MIT Artificial Intelligence Lab's Intelligent Room Project [3,5]. The Intelligent Room explores natural interaction with embedded computational systems. It has a host of computer vision and speech understanding systems that connect it to ordinary, human-level events occurring within it.

In this paper, we are concerned with the overall design and implementation of the Intelligent Room's support for natural language interactions. The main feature of this is the *recognition forest*, a linguistic data structure that simplifies incorporation of contextual information into the room's speech understanding system and allows the room's multiple applications to independently access its speech modality. Our motivation for using context is: to help manage the combinatorial explosion in processing time that followed the incorporation of natural syntactic variability into our speech recognition system; to allow for the incorporation of non-linguistic information into linguistic contextual model; to disambiguate deictic references; and to provide spoken language input to coexisting, independent applications.

In this paper, we present the recognition forest as a useful tool for creating spoken language interfaces to intelligent, interactive spaces. We will also discuss how it contributed towards satisfying the goals that shaped the design of the Intelligent Room's natural language interface, including:

1. To support unimodal speech interactions, i.e. interactions that do not tie the user to a keyboard, mouse or display.
2. To leverage off very strong notions of context inherent in room applications to allow for better speech understanding.

This material is based upon work supported by the Advanced Research Projects Agency of the Department of Defense under contract number F30602—94—C—0204, monitored through Rome Laboratory.

3. To allow multiple speech-enabled room applications to coexist without a heavyweight central controller.
4. To provide for dynamic sets of recognized utterances.
5. To employ only very shallow linguistic knowledge and representations.

The decision to minimize the amount of linguistic knowledge contained in our system was made to facilitate the room's infrastructure and application development by a wide range of people, particularly computer science undergraduates who have no formal exposure to computational linguistics. Our system needed to be accessible by all researchers in the project, regardless of their background. We believe that many of the issues discussed in this paper will remain useful when applied to systems with more sophisticated linguistic representations. Given the increasingly widespread interest in highly interactive, computational environments [7], many other designers and implementers will be faced with similar challenges, and we hope our approach will be generally useful for other systems. Over the past three decades there have been significant research efforts devoted to the development of natural language interfaces. We divide these into three distinct classes based on the modality chosen for natural language interaction. The first consists of text-only dialog systems, such as SHRDLU [12], Lunar [13], and the multitude of database query systems, such as START [8]. In these systems typewritten text is used for input and output. With the advent of improved speech recognition and synthesis, efforts were made to integrate these technologies into natural language dialog systems—our second category—such as those in [8,11,14]. However, with these the user is still expected to interact with the system at a terminal where the display of queries and recognition results are perused and then potentially disambiguated by the user. Many of these systems also make use of “push-to-speak button,” in order to allow the user to signal the speech recognizer that the next utterance is to be treated as input to the system. The final class of systems is those that operate only in the speech modality, such as SUNDIAL [1], and Jupiter [3], two telephone communication systems for answering domain specific queries. Our approach represents a significant departure from those described above. The choice of the ability to operate in a voice-only modality separates it from the text-only and mixed voice and screen systems. This departure requires an emphasis on careful planning and structuring of dialog to take advantage of speech while also overcoming some of the difficulties inherent in the speech modality. Our approach is different from that of the other voice-only language projects in that: we are not tackling applications that monopolize the user's attention or require excessive word knowledge; we allow people interacting with the Intelligent Room to readily change application contexts; we place more emphasis on speech recognition in noisy, multi-user environments where people are primarily talking to one another; we seek to plan interactions with minimum intrusiveness; and we provide an interface to multiple applications simultaneously. Our system is not intended to be task or domain specific; it is used in the same way as a keyboard and mouse—a general input device simultaneously used by many applications. It is a tool, not an end in itself.

In the next section we briefly motivate and describe the Intelligent Room as the platform and motivation for our work. Next, we present a user's perspective of the Intelligent Room. In section 4 we outline the room's computational architecture and linguistic systems. Then we give several representative linguistic

interactions illustrating the concepts we have described. Finally, we discuss limitations inherent in our approach and possible remedies.

2 The Intelligent Room

We now proceed to briefly describe the Intelligent Room as the platform for our research. More in depth discussion of the room can be found in [6] and details of its multimodal resolution are contained in [5].

The Intelligent Room is a research platform for exploring the design of intelligent environments [7]. The Intelligent Room was created to experiment with different forms of natural, multimodal human-computer interaction (HCI) during what is traditionally considered non-computational activity. It is equipped with numerous computer vision, speech and gesture recognition systems that connect it to what its inhabitants are doing and saying. The motivation for researching intelligent environments is to bring computation into the real, physical world. The goal is to allow computers to participate in human-level activities that have never previously involved computation and to allow people to interact with computational systems the way they would with other people: via gesture, voice, movement, and context.

The Intelligent Room is a space populated by computer controlled devices; these include overhead LCD projectors and displays, audio/visual multiplexers, VCRs, drapes, blinds, stereos, steerable video cameras, etc. The video cameras are used by the room's computer vision systems. These vision systems are detailed in [6], but of relevance here are the following: a person tracking system that can locate people in real-time as they move about the room; gesture recognition of both finger and laser pointing on either of the room's projected LCD displays; and a system that specifically notices when people sit down on particular pieces of furniture.

Other research in intelligent environments [2,9,10] has focused more on development of computer vision and other sensing technologies at the expense of linguistic interactions. We believe, however, that language is fundamental to having meaningful and complex interactions with these sophisticated, interactive spaces. In particular, we are interested in speech understanding systems that function more like a language modality than a voice simulation of a keyboard or mouse. Although the Intelligent Room's current linguistic systems require a great deal of development before they near this goal, we believe our approach is an extensible first approximation.

3 User Interactions

People in the Intelligent Room wear wireless lapel microphones that transmit to the speech understanding system described below. By default, the room ignores the spoken utterances of its inhabitants, which are generally directed to other people within the room. This state is known as "the room being asleep."² To obtain the room's attention, a user stops speaking for a moment and then says the word "Computer." The room immediately responds with an audible, quiet chirp from an overhead speaker to indicate it is paying attention. The user then has a two second window in which to begin speaking to the room. If the room is unable to recognize any utterances starting within that period, it silently goes back to sleep until explicitly

² The room's vision systems continue to respond to users even when it is not listening for their verbal input.

addressed again. However, if what the user says is recognized, the room responds with an audible click and then under most circumstances it returns to sleep. This hands- and eyes-free style of interaction coupled with audio feedback allows a user to ignore the room's computational presence until she explicitly needs to communicate with it. There is no need to do anything other than preface spoken utterances with the cue *Computer* to enable verbal interaction. Thus, a user can interact with the room easily, regardless of her proximity to a keyboard or monitor. Additionally, explicitly cueing the room minimizes the likelihood that extraneous speech or noise will incorrectly trigger a recognition event. Importantly, it also allows detection of non-recognition events, i.e. times when the room is not able to understanding something the user is explicitly trying to convey. In the event the room erroneously wakes up due to an incorrect recognition event, it will either go back to sleep automatically when the two second window expires or the user can explicitly tell it to "go to sleep" upon hearing the wake-up chirp.

When the room is awake, it is listening for a specific set of utterances contained in its recognition forest, a data structure described in the next section. Under appropriate circumstances, users can also freely and continuously dictate expressions to the room unconstrained by any grammar or rule set. This capability is used, for example, during information retrieval queries (such as a web search) for which it is unreasonable to expect that the room's grammars already contain the sought after phrase. In these interactions, the room repeats the final utterance back to the user to verify correct recognition. The lag time between user speech and room verification is extremely small, and this mode of interaction has proved to be quite useful provided input is short in length. We note here that the room is responsible for switching between the constrained and diction speech modes; users do not explicitly change this state.

The room can also remain awake listening for utterances. Someone intending on a prolonged series of verbal interactions can simply tell the room to "stay awake." The room continues to provide an audible click after recognized statements, but these statements no longer need to be preceded by the spoken *Computer* cue. In addition, the room can wake itself up if it expects an utterance from the user for some reason. For example, when the room asks the user a question, it will stay awake for several seconds waiting for an answer. If that period ends without a response being given, the room provides an audio timeout signal to indicate that it is going back to sleep.

4 Speech Understanding

In this section we present the Intelligent Room's speech understanding system. We begin with a discussion of the interaction between the software agents that control the operation of the room and the room's set of recognition grammars.

The Intelligent Room is controlled by a modular system of approximately 100 distinct, intercommunicating software agents that run on several networked workstations. These agents' primary task is to connect various components of the room (e.g., vision and speech recognition systems) to each other as well as to internal and external stores of information (e.g., a person locator or an information retrieval system). Essentially, these agents are intelligent *computational glue* for interconnecting all of the room's components and moving information among them.

The Intelligent Room listens for continuous speech utterances contained in a forest (or set) of multiple grammars, which we call the recognition forest. Each grammar in the recognition forest is created by one of the room's software agents (see Figure 1), which

receives notification when any utterance contained in one of its grammars is heard. Agents do not necessarily know nor need to know of any other grammars or agents. (We note for clarification that a single agent is allowed and actually encouraged to create multiple grammars.) The notification message for a recognized utterance contains a parse tree, which the agent can manipulate to determine its content. In the recognition forest, a grammar is called “active” if the room is currently listening for it and “inactive” otherwise. Active grammars are rank ordered in terms of their expected likelihood of being heard. Fundamental to our design is that all of the grammars must be constrained to highly specific contexts among which some component of the Intelligent Room is capable of distinguishing. Instead of keeping a single enormous recognition grammar active, the room collectively keeps subsets of small grammars active in parallel, given what it

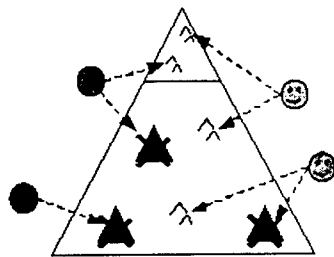


Fig. 1. Software agents creating the forest of context free recognition grammars. Each small triangle represents a grammar, and each face represents a software agent. Active grammars are lightly colored. Inactive grammars are crossed out. The uppermost, demarcated region contains universal grammars that are always active.

currently expects to hear. The key assumption here is that certain types of utterances are only likely to be said under particular circumstances. These may be related to where someone is spatially, the history of her previous interactions, how she is gesturing, what devices in the room are doing, etc. At the simplest level, this can range from the implausibility of someone saying “stop the video,” when none is playing, to more complex dependencies, such as the meaninglessness of asking “What’s the weather there?” if no geographic entity has somehow been brought to the room’s attention.

The context-dependency of these grammars is not contained within the linguistic formalism itself, which allows us to use an extremely simple representation. Rather, the room’s software agents are responsible for setting and modifying the activation states of grammars they create based on whatever information the room’s other software agents can provide about current goings on in and state of the room. (See Figure 2.) For example, if the room starts showing a video clip, the agent that controls the showing of videos activates the grammars that involve VCR operation. When the clip stops, these grammars are in turn deactivated. More interesting cues can involve the location of someone inside the room. For example, the fact that someone has moved near an interactive displayed map is sufficient reason for the room to pay increased attention for spoken utterances involving geographic information. However, until that cue is received, it seems quite reasonable not only for the room to ignore such requests but to not recognize them at all especially given the error rate of current speech recognition technology. We note there may be cases where this is inappropriate; for example, the room might alternatively need to

recognize out-of-context utterances in order to provide guidance to a user. We are investigating techniques for dealing with this, such as having the room iteratively broaden the set of active grammars and proactively offer assistance in case the user's speech is not being recognized.

When users shift to a new application context, the system lowers the relative rankings of and eventually deactivates grammars from the previous contexts according to a least-recently-used strategy. Thus, agents need not explicitly deactivate all of their grammars nor even know all appropriate circumstances for doing so.

Notions of context can also help adjust expected probabilities of utterances. Even in systems where all utterances are valid at all times, it is generally not the case that all utterances are equally likely at all times. For example, tracking context can help disambiguate the output of bigram-based speech recognition systems that return the probabilistically weighted N-best set of utterances for each recognition event. We used this scheme to process the results returned by the Galaxy System [14], which was the first speech recognition system used in the early days of the Intelligent Room.

We have found it useful to have several different notions of ongoing room context for determining which grammars are active at any given moment. However, no single agent defines "the context" but rather the context is a product of many loosely connected entities. In ranked order, these consist of the following:

1. Always active grammars – These are for low-level control and providing feedback to the room. These grammars allow direct manipulation of room state; we have found it essential for users to feel they control the room's physical infrastructure if they are to feel comfortable interacting with it. These grammars also allow the manual adjustment of various room parameters in the event of incorrect data from one of the room's input modalities.
2. Context shifting grammars – These are explicit cues for the room to change its context. They generally start new room activities and automatically lead to changes in the contents of the next two categories.
3. Current applications' grammars — These are application specific verbal interactions with the room given its current state. The room frequently modifies these grammars while it is running.
4. Previous applications' grammars — These are for interacting with previously run applications. These are particularly useful in case of inadvertent or incorrect context shift or if some device failed to respond appropriately and must be corrected via verbal interaction. Backgrounded tasks often have grammars here so they can be quickly recalled and resumed.

Agents can also modify the structure and content of an extant grammar. This ability is used currently only for inserting and deleting noun phrases to reflect newly obtained

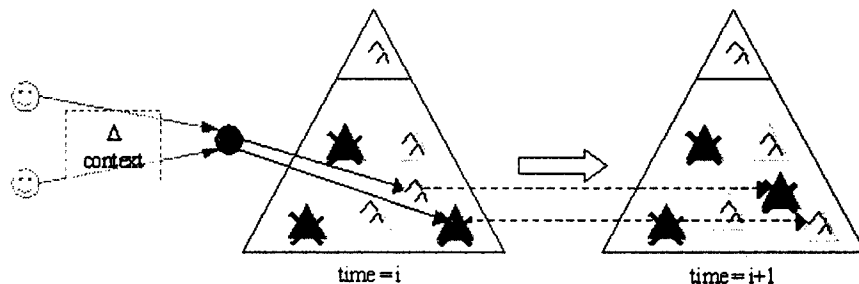


Fig. 2. A transition in the forest of grammars. An agent can activate and deactivate its grammars based on context changes in the Intelligent Room. Notification of context changes comes in the form of messages from other room agents.

information. This can be gotten from: the user verbally dictating new phrases to the room; mechanical extraction from other sources, (e.g., anchor link text in web pages); or the room applying machine learning techniques to augment its vocabulary (as discussed below).

Another advantage of the distributed nature of the grammars is individual agents can monitor their small piece of the overall context in a very simplistic fashion. For example, the VCR agent can pay attention to events only relevant to knowing whether the VCR is part of the context or not, and modify its associated grammars accordingly. This avoids the problem of clearly defining the overall context, and also eliminates the need for control logic for deciding which grammars should be active when.

Underlying Speech Technology and Computational Complexity

For processing spoken utterances, we use IBM's ViaVoice speech recognition system. ViaVoice is a commercially available system primarily used for continuous speech dictation. This, with its relatively low word accuracy for single word and short utterances, would have been an intolerable speech interface to the room. However, ViaVoice also supports explicit construction of continuous speech, context-free recognition grammars, which allow for much higher degrees of recognition accuracy. Via its Java interface, it also provides control over low-level aspects of its behavior to external applications, which makes it ideal for incorporating into other systems.

We wanted the Intelligent Room's recognition grammars to be reasonably unconstrained. In particular, we wanted to allow people to interact with the room without memorizing scripts of recognized utterances or lists of permissible syntactic constructions. However, we found that as our grammars became increasingly large, speech recognition accuracy fell correspondingly. As room grammars started to support more than a few thousand individual utterances, recognition accuracy dropped below acceptable levels. There is a clear tradeoff between making the room's recognition grammars sufficiently large so that people can express themselves somewhat freely versus making the grammars small enough so that the system runs with high accuracy and in real-time. Thus, we decided to make use of the natural context specificity in room applications so that agents could dynamically activate different subsets of grammars depending on the context of the activity within the Intelligent Room.

5 Sample Interactions

The Intelligent Room supports a variety of narrow application domains, all of which can run simultaneously. The collection of all these domains in turn gives an extremely broad and flexible 'domain' encompassing a wide range of tasks. The distributed, independent control of the recognition forest allows for this—there is no need for a centralized controller. The room's applications range from simple voice control over physical devices to more complex multimodal scenarios involving position, gesture tracking and spoken dialog. We first outline these domains and then present two applications in more detail:

1. Manual control over devices – These include the Intelligent Room's lights, blinds, drapes, VCRs, video displays, stereo components, etc.
2. Manual interaction with modal subsystems – We have found it extremely useful to have direct verbal interactions with the room's modalities. These can be used to gather information about what the room is observing, to modify internal representations of its state, or to correct a perceptual error. It is also of enormous benefit to be able to verbally interact

with the room's vision systems while developing or debugging them, because it is generally impossible to manually interact with them at a workstation while remaining in the camera's foveal areas.

3. Information access – There are many types of these interactions, including web browsing, weather reporting, accessing an online video collection, querying Haystack (a personal information manager), and the information retrieval system described below. The room also functions as a spoken language front-end to START, a natural language query database [9].
4. Presentation manager – This allows the Intelligent Room to assist in multimedia presentations and is a demonstration of the room's information management capabilities. A lab tour guide agent that uses this application for presenting a broad overview of our laboratory's research to visitors has been previously described in [5].
5. Command post – This application provides the means to test full integration of all our modal subsystems and to experiment with different techniques for performing multimodal reconciliation. It is a mock command center for planning hurricane disaster relief.

We now present two primarily linguistic interactions with the Intelligent Room. They are annotated with the changes made to the recognition forest to reflect the course of ongoing interactions. The first of these is the above-mentioned command post. It makes use of two interactive projected displays that respond to finger pointing gestures. The second interaction is a primarily unimodal dialog that allows users to interactively refine a document retrieval query.

In the following interaction the user is attempting to plan disaster relief for a hurricane in the Virgin Islands.

Command Post:

User: *"Computer, stay awake."*

[The room will now listen for utterances without requiring they be prefaced by the word Computer.]

[The person's approach of projected displays causes the room to pay attention to statements involving them. This is illustrated in Figure 3.]

User: *"Show me the Virgin Islands."*

Room: *"I'm showing the map on the display next to you."*

[Room shows map on video display closest to the user.]

[Room activates grammars associated with the map.]

[User now points with his finger at St. Thomas.]

[Room adds nouns (such as city names) relating to St. Thomas to active grammars.]

[Room now considers St. Thomas the default geographical entity.]

User: *"Zoom in."*

[Room zooms in to St. Thomas.]

User: *"How far away is Hurricane Marilyn?"*

Room: *"The distance between Hurricane Marilyn and the city of Charlotte Amalie located in St. Thomas is 145 miles."*

User: *"Where's the nearest disaster field office?"*

[Room highlights them on the map.]

Room: *"The St. Thomas DFO is located one mile outside of Charlotte Amalie. There is a new weather forecast available. Do you want to see it?"*

[Room activates a grammar containing possible expected responses for this question.]

User: *"Yes, show me the satellite image."*

[Room displays image on adjacent projected display.]

[Additional weather grammars are now activated.]

The following interaction is an example of the user dynamically switching contexts while in the midst of interacting with the Intelligent Room. The room has an information retrieval system that helps users refine their queries by asking them questions that maximally disambiguate it. These questions are determined by

searching a document tree obtained from the Alta Vista web search engine for phrases with the highest entropy levels.

Information retrieval:

User: *"Computer, I need information."*

[Because user didn't specify a topic, room will ask for it to be dictated.]

Room: *"About what? Please tell me."*

User: *"Software agents."*

Room: *"Is software agents correct?"*

User: *"Yes."*

[Room will now remember this phrase in the context of information retrieval to allow recognition of it in the future. Other agents might eavesdrop and also add this word for their own purposes, in an attempt to anticipate the user's eventual needs.]

Room: *"Thank you. Please wait a moment...Are you interested in HCI, Research & Indexes, or General AI?"*

[Room activates a grammar of expected responses based on these topics.]

User: *"I'm interested in HCI."*

[User can respond with continuous speech even though the phrase "H C I" was not in any room grammar before this interaction, since it was just added.]

[Above constructed response grammar is deactivated.]

Room: *"Okay, are any of these documents of interest? I will put them on the left display."*

[Room displays document list on wall.]

[Room activates a response grammar that incorporates the titles of displayed documents.]

User: *"Please move them to the display near the window."*

[Room does so]

The above examples illustrate how the room manages the recognition forest via data from both its perception subsystems and from expectations of what the user is likely to say in a given situation. This allows the room to approximate natural linguistic interactions with the user. Of course, it is still somewhat stilted, and the user cannot make complete non-sequiturs. Overall, however, the main problem we have is convincing the user to push the limits and speak naturally, rather than linguistically downsizing and second-guessing the room's capabilities, but this fault is prevalent in most, if not all, current speech understanding systems.

6 Achieving Design Goals

We review the design goals presented in the first section, considering not only how well they were achieved, but possible remedies for where our approach was unsuccessful.

A language modality

A key aspect of the Intelligent Room is for an occupant to have full access to all of the room's computational power regardless of where in the room she is. She should not need to type at a particular keyboard nor interact with a particular display to interact with the room, and in fact, the room does not have a keyboard or mouse within it.

Therefore, we decided that in many circumstances speech interactions had to be unimodal. It could not be the case that the room would need to display candidates for recognized utterances, thereby allowing the user to select among them or to disambiguate didactic references. This is contrasted with the approach of [11,14] where users provide critical feedback during the recognition process via a graphical user interface. Our approach of having the room ask the user when it is unsure of something can be somewhat intrusive, but it is certainly no more so than a graphical interface.

Context sensitivity

We wanted to make use of context in terms of both the room and user's states to be able to both resolve deictic references and control sets of possible utterances and who should receive those utterances. As in most speech understanding efforts, we wanted to support some measure of natural syntactic variability on the part of a person interacting with room. Our intent was to leverage off the well-defined notions of context inherent in the Intelligent Room's application domains to keep the total active grammar size small at any given time. This can be enormously frustrating if the room inappropriately deactivates a grammar to which the user would still like to refer. We are currently exploring techniques for dealing with this, such as reprocessing the spoken audio signal under an iteratively broadened set of grammars.

Non-static recognition sets

We sought to avoid limiting the room to a static set of recognition grammars. It would not have been reasonable to suppose that we could determine everything in advance users would want to say, and it would have made routine tasks like information retrieval difficult, if not impossible.

The ability of agents to change grammars on the fly has proved to be extremely useful, in applications such ranging from web browsing, where link anchor text is captured, to information retrieval, which typically involves an iterative query refinement process. The ability of the room to incorporate user-dictated noun-phrases into its recognition grammars is one of the capabilities that most impresses new users.

On a larger scale, adding agents should be easy. By having individual agents control their own grammars and activation states, agents can indeed be added quickly and without worry as to their disrupting other agents' interfaces.

Simplicity

Finally, we were also interested in employing very shallow linguistic knowledge during implementation to minimize the knowledge engineering problem. Given that new room applications are being created on a regular basis, it is not possible to build carefully handcrafted linguistic models of expected input. Speech orientated agents have proliferated markedly since our system came up, to a point of fault. Speech is such a natural and easy modality that the temptation to solve all problems with it has distracted us from really striving for a multi-modal system that pays attention to harder to discern inputs, such as gesture.

Future work on the system includes incorporating a machine learning mechanism into the recognition forest so that it can learn the probabilities of individual grammars be used in particular application contexts. We are also interested in learning the transition probabilities among the grammars, to better predict activation states without requiring explicit action be taken by the room's software agents.

7 References

- [1] Andry, F., Fraser, N.M., McGlashan, S., Thornton, S., and Youd, J. Making DATR Work for Speech: Lexicon Compilation in SUNDIAL. *Computational Linguistics*, 18(3), pp. 245-267. 1992.
- [2] Bobick, A.; Intille, S.; Davis, J.; Baird, F.; Pinhanez, C.; Campbell, L.; Ivanov, Y.; Schütte, A.; and Wilson, A. Design Decisions for Interactive Environments: Evaluating the KidsRoom. *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
- [3] Chung, G., and Seneff, S. Improvements in Speech Understanding Accuracy Through the Integration of Hierarchical Linguistic, Prosodic and Phonological Constraints in the Jupiter Domain, *Proc. ICSLP '98*, November 1998.
- [4] Coen, M. A Prototype Intelligent Environment. In Streitz, N., et al. (Eds.), *Cooperative Buildings - Integrating Information, Organization, and Architecture*. *Proceedings of the First International Workshop on Cooperative Buildings (CoBuild'98)*. *Lecture Notes in Computer Science*. Springer: Heidelberg. 1998.
- [5] Coen, M. Building Brains for Rooms: Designing Distributed Software Agents. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*. (IAAI'97). Providence, R.I. 1997.

- [6] Coen, M. Design Principles for Intelligent Environments. In Proceedings of The Fifteenth National Conference on Artificial Intelligence. (AAAI98). Madison, Wisconsin. 1998.
- [7] Coen, M. (ed.) Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments. AAAI TR SS-98-02. 1998.
- [8] Federico, M. and Vernesoni, F. "A speech understanding architecture for an information query system". Proceedings of EUROSPEECH 95, Madrid, Spain, 1995.
- [9] Katz, B.. Using English for Indexing and Retrieving. In Artificial Intelligence at MIT: Expanding Frontiers. Winston, P.; and Shellard, S. (editors). MIT Press, Cambridge, MA. Volume 1. 1990.
- [10] Mozer, M. The Neural Network House: An Environment that Adapts to its Inhabitants. Proceedings of the AAAI 1998 Spring Symposium on Intelligent Environments. AAAI TR SS-98-02. 1998.
- [11] Stock, O., Carenini, G., Cecconi, F., Franconi, E., Lavelli, Magnini, B., Pianesi, F., Ponzi, M., Samek-Lodovici, V., and Strapparava, C.. ALFRESCO: Enjoying the Combination of Natural Language Processing and Hypermedia for Information Exploration. In Maybury, M. (ed.), Intelligent Multimedia Interfaces. The MIT Press, pp. 197-224.
- [12] Winograd, T. Five lectures on artificial intelligence. Technical Report, Stanford-CS, Number AI Memo 246, Stanford University, 1974.
- [13] Woods, W., Kaplan, R., and Nash-Weber, B. The Lunar Sciences Natural Language Information System: Final Report. Technical Report, Bolt Beranek and Newman, Number 2378, June 1972.
- [14] Zue, V. Human Computer Interactions Using Language Based Technology. IEEE International Symposium on Speech, Image Processing & Neural Networks. Hong Kong. 1994.

Meeting the Computational Needs of Intelligent Environments: The Metaglue System

Michael H. Coen, Brenton Phillips, Nimrod Warshawsky, Luke Weisman, Stephen Peters, and Peter Finin.

MIT Artificial Intelligence Lab
545 Technology Square
Cambridge, MA 02139
mhcoen@ai.mit.edu

Abstract. Intelligent Environments (IEs) have specific computational properties that generally distinguish them from other computational systems. They have large numbers of hardware and software components that need to be interconnected. Their infrastructures tend to be highly distributed, reflecting both the distributed nature of the real world and the IEs' need for large amounts of computational power. They also tend to be highly dynamic and require reconfiguration and resource management on the fly as their components and inhabitants change, and as they adjust their operation to suit the learned preferences of their users. Because IEs generally have multimodal interfaces, they also usually have high degrees of parallelism for resolving multiple, simultaneous events. Finally, debugging IEs present unique challenges to their creators, not only because of their distributed parallelism, but also because of the difficulty of pinning down their "state" in a formal computational sense. This paper describes Metaglue, an extension to the Java programming language for building software agent systems for controlling Intelligent Environments that has been specifically designed to address these needs. Metaglue has been developed as part of the MIT Artificial Intelligence Lab's Intelligent Room Project, which has spent the past four years designing Intelligent Environments for research in Human-Computer Interaction.

Introduction

Research on highly interactive spaces, generally known as Intelligent Environments, has become quite popular recently. Although their precise applications, perceptual technologies, and control architectures vary a great deal from project to project, the *raison d'être* of these systems are generally quite similar. They are aimed at allowing computational systems to understand people on our own terms, frequently while we are busy with activities that have never before involved computation. IEs seek to connect computational systems to the real world around them and the people who inhabit it.

This paper presents what we believe are general computational properties and requirements for IEs, based on our experience over the past four years with the Intelligent Room Project at the MIT Artificial Intelligence Lab. Although many of the published descriptions of IEs [4] differ in their particulars, it is clear that we have not been alone in confronting some of

This material is based upon work supported by the Advanced Research Projects Agency of the Department of Defense under contract number F30602—94—C—0204, monitored through Rome Laboratory.

the frustrating aspects of engineering these complex systems. Based on this experience, we have developed Metaglua, a specialized language for building systems of interactive, distributed computations, which are at the heart of so many IEs. Metaglua, an extension to the Java programming language, provides linguistic primitives that address the specific computational requirements of intelligent environments. These include the need to: interconnect and manage large numbers of disparate hardware and software components; control assemblies of interacting software agents *en masse*; operate in real-time; dynamically add and subtract components to a running system without interrupting its operation; change/upgrade components without taking down the system; control allocation of resources; and provide a means to capture persistent state information.

Metaglua is necessary because traditional programming languages (such as C, Java, and Lisp) do not provide support for coping with these issues. There are currently several other research systems for creating assemblies of software agents [7,8,9], which provide low-level functionality, e.g., support for mobile agents and directory services. These features are necessary but not sufficient. Because Metaglua provides high-level tools specifically relevant to creating software controllers for IEs, we hope to make it available for more widespread use by the IE community.

Much of our discussion will focus on *Hal*, our most recently constructed Intelligent Room [3,5], where approximately 100 Metaglua agents control Hal and interconnect its components. However, we believe the issues raised here extend beyond the particulars of Hal and are important for a wide range of intelligent environments.

Hal is a small room within our lab and is equipped with microphones, seven video cameras, and a variety of audio-visual output devices that it can directly control. Hal was designed to explore a wide range of interactions involving futuristic residential spaces – stressing quality of life – and commercial spaces – stressing information management. We have therefore created applications in Hal that support entertainment, teleconferencing, business meetings, military command post scenarios, and information retrieval.

Next, we expand on our list of computational properties for IEs and examine the reasons behind them. We then discuss design considerations of the Metaglua system, and how it specifically addresses the perceived needs of IEs. In this, we directly trace how the issues raised in the next section are satisfied by capabilities incorporated into Metaglua. Finally, we close with an evaluation of the Metaglua system and directions for future research.

Computational Properties of Intelligent Environments

Intelligent Environments by and large share a number of computational properties due to commonalities in how they internally function and externally interact with their users. Of course, not every IE will identically share all of these characteristics, but we believe examining them even briefly makes concrete many of the issues that IE designers are faced with and should address directly. We not only hope to further discussion on these issues in the IE community, but to motivate the development of other general purpose tools such as Metaglua.

We note that when multiple people are allowed to interact simultaneously with a single IE, many of the issues discussed below are greatly exacerbated. Space limitations preclude addressing this issue in detail.

Distributed, modular systems need computational glue

Intelligent Environments contain a multitude of subsystems comprising their perceptual interfaces, software applications, hardware device connections, and mechanisms for internal control. Even though each IE is created in its own way for its own purpose, IEs are generally built out of similar components.

Thus, IEs require some way to *glue* all of these components together and coordinate their interactions. These components also generally cannot co-exist on a single computer, due to hardware constraints and the need for environments to respond in real-time to their users. It is not uncommon for individual computer vision or speech understanding applications to consume the resources of an entire workstation, and there is no reason to believe this exclusivity will be diminished as processor speeds increase in the future.

Many of these systems perform progressive real-time searches that naturally generalize to consume all increases in available computational power.

Frequently, these components, either off the shelf or research programs in their own right, are not designed to work together, so not only must they be connected, but there needs to be some way of expressing the "logic" of this interconnection. In other words, inter-component connections are not merely protocols, but must also contain the explicit knowledge of how to use these protocols. Thus, viewing the connections simply as Application Programming Interfaces (APIs) is insufficient. For example, consider connecting a speech recognition system and a web browser, so that users can navigate links by speaking the text contained in them. Here, the computational *glue* would include a mechanism that dynamically updates a recognition grammar with the link text whenever the user goes to a new web page; simply having APIs to both of these applications is necessary but not sufficient.

More generally, enormous amounts of control code go into building IEs, much of it dealing with how connections among its pieces should be managed. (See [1,2].)

Resource management is essential

Interactions among system components in an IE can be exceedingly complex. Resources, such as video displays or computational power, can be scarce and need to be shared among different applications. For example, in Hal, multiple computer vision systems share individual video cameras because they are a relatively expensive resource [6]. Conflicts can occur when multiple applications in an IE all want to display information on a single video display or speak to the user simultaneously. One of the largest surprises while developing Hal was that even seemingly simple issues, such as displaying a video, have wide spread repercussions on other parts of the running system because their resources are pulled out from under them. We discuss this at greater length in the next section. Finally, even in an environment with ample resources for a single user, conflicts can unknowingly arise when multiple people attempt to interact with it simultaneously.

Thus, IEs need sophisticated resource management capabilities, particularly to let them scale properly as new applications and capabilities are added.

Configurations change dynamically

IEs can be highly dynamic systems. In the prescient words of Weiser – referring to Ubiquitous Computing but equally relevant to IEs – “*New software ... may be needed at any time, and you’ll never be able to shut off everything in the room at once ... functionality may shrink and grow to fit dynamically changing needs*” [12]. People may come and go at will, bringing with them devices such as PDAs that temporarily connect with an IEs existing computational infrastructure.

In a developing system such as Hal, new hardware and software components are incorporated on a regular basis. It should be possible to add them to a running system without restarting unrelated components. In fact, under many circumstances, new permanent components should dynamically integrate themselves into an IE without interrupting its operation at all.

Even within the confines of a static IE, users may readily switch between different aspects of its functionality. For example Hal supports teleconferencing and information management applications and users readily switching between the two is quite natural; often during meetings the need arises to get more information about something. These “context” changes can have far reaching effects. For example, an IE may need to simultaneously start new underlying applications, activate different speech recognition grammars, and modify the configurations of other perceptual systems.

State is precious

Not only may new components need to be incorporated into a running IE, but pieces of a running system may need to be reloaded into it as well. As with any other software engineering effort, creating IEs require an iterative edit-recompile-run process while testing new features and eliminating bugs. However, if the entire system had to be restarted each time one of its components changed, development would be prohibitively time-consuming. Our Hal environment has literally dozens of hardware and software components connected to approximately 100 Metaglug agents. Having to bring this system completely down to modify it would long ago have made further development a cause of endless frustration.

Furthermore, IEs acquire state through interactions with users. Attempting to trace a bug by forcing a person to repeat a sequence of interactions that may have spanned several hours would be outrageous. To exacerbate the problem, state is acquired not only through human interactions, but also through any activities Hal has engaged in, such as information retrieval. A weather report or CNN headline that caused Hal to take some action may have long since changed and is not recoverable.

The most critical part of Hal’s state comes from information it learns while observing its users. Hal has several machine learning systems for learning about users’ preferences and activities. These systems have no straightforward way to unlearn and return to a previous coherent state. Checkpointing in the style of reliable transaction systems can partially ameliorate these problems with respect to the local state of individual components. However, when IEs are asynchronous and distributed, repeating a particular *global* state can be, practically speaking, impossible to achieve. (One technique we have been investigating is allowing an IE to essentially simulate itself by replaying previously observed and recorded events.)

Thus, there is a clear need for an IE's software architecture to permit a kind of dynamism rare in conventional computational systems. We would like to stop, modify, and reload components of a running system and have them reintegrate into the overall computation with as much of their state intact as possible.

IEs model the parallelism of the real world

Supporting natural human computer interaction requires that IEs have some handle on multiple ways that a user may interact with them. People speak, gesture, move, and emote simultaneously, and IEs need to have some capacity to cope with this, even on the part of a single user. This is not to say they need to understand the full range of human discourse to be useful. Far from it, IEs that consistently – and most importantly, predictably – understand a small subset of interactions are far preferable from an HCI perspective to ones that always leave users guessing if some particular input will be understood.

Nevertheless, IEs generally have multimodal interfaces which requires they have sufficient parallelism for resolving multiple, simultaneous events. For example, if a user walks to a displayed map, points somewhere and says, "What's the weather here today?" and then immediately walks away, the system must be able to discover where they were pointing when they said the word "here." Dealing with multiple users simultaneously again simply exacerbates the problem.

Thus, IEs need at least as much parallelism as the phenomena they are trying to understand. In fact they may need a great deal more for background processing, which leads to the next item.

Real-time response

It almost goes without saying that IEs need to be responsive to their users. Particularly due to the fact that many IEs do not have traditional computer monitors so users can get a handle of the system's inner activity, it is astonishingly frustrating when an IE does not respond quickly to user input. This is another point supporting the basic need for parallel architectures in an IE. The parts of the system that acknowledge and react to users must be immediately responsive even if other parts of the system, for example, in the midst of processing an information retrieval query, require more time to respond.

This also requires that the mechanism for interconnecting an IE's components and processing their data be able to keep up with the underlying external systems. For example, in Hal, five C-language-based computer vision systems, each producing several hundred dimensional data vectors at a rate of up to 30 a second, all connect to a Metaglove-based visual event classification system which must process all this data in real-time.

Debugging is difficult

Independently of IEs, debugging distributed, asynchronous systems can be a nightmare. If some high-level system event fails to occur, determining which component is to blame is usually a long, involved process. Furthermore, understanding the operation of distributed, loosely coupled components running in parallel – as does the controller for an IE – where different serializations can have different system-wide effects, is best, but rarely successfully, avoided. In an IE, this problem is made all the worse by the presence of many, sometimes exotic, hardware components, such as video multiplexers, that

themselves have internal state that may only be imperfectly modeled in their software drivers.

Good software engineering practices go a long way towards dealing with this problem, but a more comprehensive solution would require the development of new types of debugging strategies. (In the next section, we see that Metaglua only makes the most preliminary efforts in this direction.)

Metaglua

We first discuss the design of Metaglua from a programming language perspective, to give potential users a sense of what it would be like to work with it. We then proceed to illustrate how particular features in Metaglua address many of the computational requirements for IEs discussed in the previous section. By necessity, this section is intended only to sketch and motivate the capabilities of the Metaglua system and should not be viewed as a complete description of the language. More detail about Metaglua's internals can be found in [10,11]. (Some examples in this section require cursory knowledge of the Java programming language.)

The Design

Metaglua is an extension to the Java programming language that introduces a new *Agent* class. By extending this class, user-written agents can access the special Metaglua methods discussed below. Metaglua has a post-compiler, which is run over Java-compiled class files to generate new Metaglua agents. Metaglua also includes a runtime platform, called the *Metaglua Virtual Machine*, on which its agents run. The overhead added by this infrastructure to standard Java programs that are turned into Metaglua agents is negligible.

Our goal with Metaglua was to add a very small number of primitives to the Java language to make it easy to write agents. Method invocations between agents, even if they are on different workstations, look exactly like local method calls in Java. Thus, Metaglua agents, minus the few Metaglua-specific primitives, look almost exactly like ordinary Java programs. This makes it easy to transform regular Java source files into Metaglua agents, which enormously adds to Metaglua's value as computational glue. We call the process of transforming previously existing programs into agents *wrapping*.

Almost as much time has gone into formulating Metaglua's semantics as to programming the system itself. We sought to provide a focused set of primitives for managing systems of distributed, interacting agents and to avoid the temptation of creeping featurism. By stressing simplicity, anyone proficient in Java can pick up Metaglua very quickly, and the small number of new primitives makes it easy to learn, remember, and use the system.

In the remainder of this discussion, it will be helpful to keep in mind that running a Metaglua system first involves starting Metaglua Virtual Machines on all the computers that are involved. Our machines are generally configured to start these when they are booted.³

The Capabilities

Metaglua offers the following capabilities, each of which we will address in turn:

³ For reference, this has the computational overhead of running one Java Virtual Machine, which is close to unnoticeable on modern Pentium-based systems.

1. Configuration management
2. Establish *and maintain* the configuration each agent specifies
3. Establish communication channels between agents
4. Maintain agent state
5. Introduce and modify agents in a running system
6. Manage shared resources
7. Event broadcasting
8. Support for debugging

Metagluе has a powerful naming scheme for agents that is beyond the scope of this document. We will use here the simplest form of it, the name of the Interface file of an agent, which is in the Java class package format, e.g., an agent for controlling a television might be referenced by *device.Television*.

1. Configuration Management

Metagluе has an internal SQL database for managing information about agent's modifiable parameters (called Attributes), storing their internal persistent state, and giving agents fast, powerful database access.⁴

Attributes contain information that might otherwise be hardcoded inside agents and difficult to modify, for example, what workstation the agent needs to run on or parameters that affect its operation. Metagluе has a web-based interface for modifying Attributes, which can be changed even while an agent is running. This is one of Metagluе's mechanisms for both configuring a system of agents and interacting with it while it is operational.

This is code an agent would use to get its *location* Attribute from Metagluе's built in database:

```
Attribute location = new Attribute("location");
System.out.println("I run on " + location.getValue());
```

2. Agent Configurations

Metagluе agents can specify particular requirements that the system must insure are satisfied before they are willing to run. These can include the name of a particular computer they must be run on; specifications for particular types of hardware they require access to; and more abstract capabilities that must be available on whichever Metagluе Virtual Machine (MVM) they are run on. These are expressed with the *tiedTo()* primitive, as in:

```
tiedTo(location.getValue());
tiedTo(capability.FrameGrabber);
tiedTo(device.Television);
```

If an agent is started on an MVM that does not meet its stated requirements, Metagluе will move it somewhere else that does. If Metagluе needs to restart an agent due to localized hardware or software failure, it will attempt to find an alternative MVM on which to run the agent that also satisfies these requirements. (See item 5.)

3. Agent Connections

⁴ The use of an internal database helps enormously in dealing with Java's poor file access capabilities. Agents use the database rather than store information in files, which is particularly important because agents can move to different machines while they are running.

Because Metaglué is intended as computational glue, it needs to establish paths of communication between agents, regardless of where they are running. The `reliesOn()` primitive connects agent with capabilities they can request services from. For example, to use Hal's speech synthesizer, an agent might contain:

```
Agent speechSynthesizer = reliesOn(speech.Synthesizer);
speechSynthesizer.say("Hello! How are you?");
```

The reference to `speech.Synthesizer` refers to an abstract capability, not a particular agent. Because agents refer to each other by their capabilities and not directly by name, new agents can easily be added to the system that implement preexisting capabilities without modifying any of the agents that will make use of them. (A more sophisticated way of obtaining capabilities is described in the Metaglué resource manager below.)

Metaglué will try to locate an agent that provides the requested capability on any of the system's computers' MVMs and return a reference to it to the caller. Metaglué has an internal directory called a *Catalog* that it uses to find agents once they are started. Metaglué agents automatically register their capabilities with the Catalog when they are run.

If no agent offering this capability is found, Metaglué automatically starts one and invisibly insure that it continues running as long as it is in use.

The `reliesOn()` primitive makes it very easy to interconnect agents with a single line of code. Reliance is also a transitive operation. For example, starting the single Hal *demo* agent results in the entire Hal system being loaded because of their chain of reliances. Also, because they have been formally relied upon, Metaglué will attempt to insure that they continue running indefinitely, as discussed below.

4. Agent State

Agents can use Metaglué's `freeze()` and `defrost()` primitives to store and subsequently retrieve their fields from Metaglué's internal SQL database. The standard way of doing this is having an agent directly freeze its state when it is shutting down (or at any other appropriate time), and subsequently defrost itself in its constructor the next time it is started up. Other aspects of an agent's state, e.g., its connections to other agents, are internally maintained by Metaglué and generally do not need to be specifically managed by the agents themselves.

As of yet, we do not have a well-defined schema for capturing the global state of all the agents in a running Metaglué system.

5. Modifying a running system

Metaglué will attempt to keep a running system of agents alive. If an agent is manually stopped, for example, during debugging, the agents that rely upon it will by default simply wait for it to return in the event they need to access it. When the user restarts the agent, it will reload its frozen state and simply pick up where it left off, first dealing with any pending requests from other agents.

It is also possible to programmatically specify actions, other than simply waiting, that an agent can do if someone it relies upon is stopped. For example, it might temporarily switch to another active agent that offers the same capabilities. Metaglué's resource management can help the system in the event capabilities must be shared due to part of the system being unavailable.

If an agent dies because of unanticipated hardware or software failure, Metaglue will try to restart it automatically, switching to another MVM if necessary, but still meeting the agent's required configuration if possible. It is important to note that unanticipated crashes may cause state information to be lost, and agents who are sensitive to this should refuse to be automatically restarted. For example, an agent that controls Hal's lighting systems may not know whether the lights are on or off if it is restarted after a crash because the state information it defrosts may be inaccurate. However, in that case, it can simply ask Hal's vision agents whether they can see anything, and thereby determine the state of the lights in the room. An agent running part of an application, however, could be started out of sync with the rest of the system, and manual intervention may be required to correct the problem.

Interestingly, the Metaglue system is itself recursively constructed out of a special set of Metaglue agents. These agents have the full functionality of the system available to them, so they can for example, use Hal's speech synthesizer to let users know of internal problems in the system and ask for help resolving them.

6. Managing shared resources

Among the largest and most complex systems in Metaglue is its resource manager. Before it existed, agents in Hal simply grabbed the resources they needed and configured them at will. That a resource management system was necessary became apparent when Hal developed to the point that its multiple applications conflicted with one another and could no longer be run simultaneously. Additionally, for an agent to simply rely on the resources it wants to use, it has to know both what resources exist and are available. As devices and other agents dynamically come and go in the system, this means that every agent would need to keep track of the different resources that offer the sets of capabilities it needs. We discussed above that agents may temporarily make use of substitutes if the agents they generally rely upon are unavailable. Where should that knowledge of possible alternatives come from?

The resource system in Metaglue allows agents to request functionality at a very high-level, without being concerned with how it is provided or resolving resource conflicts among themselves. Metaglue has a hierarchical set of *dealer* agents that are responsible for distributing resources to the rest of the system. There are a wide assortment of different prototype dealer agents available, each of which has its own specified internal logic for performing allocation, substitution, etc. These dealers can be used directly by Metaglue programmers or extended to customize their operation.

Dealers not only give out resources, but they can withdraw previously allocated ones to redistribute them, based on any of several priority and fairness schemes. For example, there are dealers in Hal for allocating televisions, video projectors, and displays in general. An agent must use the dealers to gain access to any of these. If a higher priority agent needs access to a particular display, it will be temporarily withdrawn from the agent who has allocated it until it becomes available again, at which point it will be given back.

7. Event Broadcasting

In addition to agents making direct requests of one another through method calls, Metaglué agents can pass messages among themselves. Agents can register with other agents, including the Metaglué system agents, to find out about events going on in the system. For example, an agent in Hal interested in greeting people by name when they walk inside the room, simply registers with the vision-based Entry agent to request messages about *entrance* events where the identity of the person can be determined. When these events occur, it receives a message and uses the agent offering speech synthesis capability to say hello to them.

We also use event broadcasts to notify groups of agents about context shifts in room applications to dynamically and uniformly modify Hal's behavior.

8. Debugging

Metaglué has a graphical interface for examining a running system of agents called the Catalog monitor. It displays all running agents and their reliance interconnections. Clicking on an agent brings up a window in a read-eval-print loop, in which users can interactively call the agent's methods.

Metaglué also has a logging facility to manage and centralize agents' textual output. This can be useful for programmers to watch the output of particular agents without worrying about where they are running or where their output streams are being printed.

We have found these capabilities quite useful, but would still prefer source level debugging of remote agents, a dynamic object browser, and ways to set breakpoints over whole groups of agents simultaneously. At least some of these capabilities promise to be available shortly in commercial Java products and we hope to make use of them during Hal's continued development.

Discussion

Evaluating the merits of a programming language can defy objectivity. Nonetheless, Metaglué has been extraordinarily useful in building Hal, and it is highly doubtful Hal would have reached its present level of development with it. Metaglué is a very stable system, and we have left large assemblies of agents running for up to a week without any difficulties. (These systems were eventually stopped for development purposes.)

We now reexamine each of the previously mentioned properties of IEs in the context of Metaglué.

Distributed, modular systems need computational glue

Metaglué not only provides a channel to interconnect Hal's components, but it also provides the means to build applications for Hal. Rather than use a special communication mechanism, such as CORBA or KQML, separate from the system's internal controller, Metaglué allows us to reduce the amount of infrastructure by providing for both communication and control with a much lighter-weight system.

Resource management is essential

The resource management system in Metaglué not only offers a wide range of default behaviors, but it is easily customizable through Java's class extension mechanism. It is among Metaglué's most developed systems and we are in the process of incorporating it into the applications that predated it.

Configurations change dynamically

Metaglugue offers several mechanisms for coping with dynamically changing systems. The Configuration Manager and Attribute system allows users to reconfigure agents while they are running. The fact that agents refer to each other by abstract capabilities means that new agents can be incorporated into a running system without modifying any of the agents that might rely upon them. Metaglugue's ability to start and stop agents while leaving the rest of the system running allows us to dynamically "hotswap" components of a running computation. Finally, by substituting new resource managers into a running system, new functionality can be added that previously no agents were aware of.

State is precious

Metaglugue offers support for persistent local state in agents via its freeze and defrost mechanisms. Notions of global state, however, remain illusive concepts.

IEs model the parallelism of the real world

Java is inherently multithreaded, which Metaglugue inherits from it. Metaglugue's resource management allows agents running in parallel to avoid conflicting with one another. The event broadcast mechanism also simplifies communication among interacting groups of software agents running simultaneously.

Real-time response

The amount of overhead Metaglugue adds to Java is minimal. Our avoidance of heavyweight, specialized communication packages allows Metaglugue agents to essentially run as quickly as Java's Remote Method Invocation system. Metaglugue is now incorporated into "tight" loops in our code, along the most processor intensive critical paths, such as in our computer vision systems. The development of JIT compilers for Java has enormously reduced our need to place perceptory components of our system into external C-language libraries.

Debugging is difficult

Metaglugue certainly makes it possible to debug distributed agents systems, but one can hope for more. There is reason to believe the Java community as a whole shares some of this interest and will take steps in this direction.

Future directions

We are presently incorporating an expert system into Metaglugue to allow more sophisticated reasoning about system configuration and resource management. We are also creating a machine learning extension to Metaglugue, which will incorporate pieces of the system described in [6].

References

1. Bobick, A.; Intille, S.; Davis, J.; Baird, F.; Pinhanez, C.; Campbell, L.; Ivanov, Y.; Schütte, A.; and Wilson, A. Design Decisions for Interactive Environments: Evaluating the KidsRoom. *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
2. Coen, M. Building Brains for Rooms: Designing Distributed Software Agents. In *Proceedings of the Ninth Conference on Innovative Applications of Artificial Intelligence*. (IAAI97). Providence, R.I. 1997.
3. Coen, M. Design Principles for Intelligent Environments. In *Proceedings of The Fifteenth National Conference on Artificial Intelligence*. (AAAI98). Madison, Wisconsin. 1998.
4. Coen, M. (ed.) *Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments*. AAAI TR SS-98-02. 1998.
5. Coen, M. The Future Of Human-Computer Interaction or How I learned to stop worrying and love My Intelligent Room. *IEEE Intelligent Systems*. March/April. 1999.
6. Coen, M., and Wilson, K. Learning Spatial Event Models from Multiple-Camera Perspectives in an Intelligent Room. *In submission*.

7. General Magic. *Odyssey (Beta 2) Agent System Documentation*. <http://www.genmagic.com/agents>.
8. Lange, D. and Oshima, M. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley, 1999.
9. ObjectSpace, Inc. *ObjectSpace Voyager Core Package Technical Overview (Version 1.0)*. December 1997. <http://www.objectspace.com/voyager/whitepapers>.
10. Phillips, B. *Metaglu: A Programming Language for Multi-Agent Systems*. M.Eng. Thesis. Massachusetts Institute of Technology, 1999.
11. Warshawsky, N. *Extending the Metaglu Multi-Agent System*. M.Eng. Thesis. Massachusetts Institute of Technology, 1999.
12. Weiser, M. The Computer for the 21st Century. *Scientific American*, pp. 94-10, September 1991.

***MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)***

*The advancement and application of Information Systems Science
and Technology to meet Air Force unique requirements for
Information Dominance and its transition to aerospace systems to
meet Air Force needs.*